目　　录

前言
第1章　运筹学概述 ………………… 1
1.1　运筹学的特点及其应用 ………… 2
1.1.1　朴素运筹学思想及其深刻内涵 ………………… 2
1.1.2　运筹学研究的工作步骤 ………………… 2
1.2　运筹学建模 ………………… 4
1.2.1　运筹学建模的一般思路 ………………… 4
1.2.2　运筹学模型的评价 ………… 5
1.2.3　运筹学模型的求解 ………… 6
1.3　基本概念和符号 ………………… 8
1.3.1　空间与向量 ………………… 8
1.3.2　梯度向量与 Hesse 矩阵 ………………… 8
1.3.3　点和方向 ………………… 9
第2章　基本概念和基本理论 ………………… 10
2.1　基本概念 ………………… 10
2.2　经典优化算法 ………………… 11
2.2.1　线性最优化 ………………… 11
2.2.2　非线性最优化 ………………… 12
2.3　启发式算法 ………………… 13
2.4　全局最优与计算复杂性 ………… 15
2.5　计算误差理论 ………………… 17
2.5.1　误差产生的原因和形式 ………………… 17
2.5.2　误差处理的几种方法 ………… 17
2.5.3　病态函数的判别 ………………… 19
2.5.4　算法的稳定性 ………………… 19
第3章　MATLAB 基本介绍 ……… 21
3.1　MATLAB 的发展历程和影响 ………………… 21
3.2　MATLAB 界面介绍 ………………… 22
3.3　MATLAB 操作介绍 ………………… 23
3.4　M 文件函数 ………………… 42
3.5　Excel-Link ………………… 50
第4章　优化算法的基本结构 ………………… 55
4.1　常用的算法搜索结构 ………………… 55
4.1.1　收敛性的概念 ………………… 55
4.1.2　收敛准则（停止条件） …… 56
4.1.3　收敛速度 ………………… 56
4.1.4　线性搜索算法 ………………… 57
4.1.5　二次模型 ………………… 57
4.1.6　下降算法模型 ………………… 58
4.2　一维搜索算法 ………………… 59
4.2.1　黄金分割法（精确一维搜索） ………………… 60
4.2.2　进退法 ………………… 62
4.2.3　沃尔夫法 ………………… 66
4.3　MATLAB 函数 Fminbnd ………… 70
第5章　线性规划 ………………… 72
5.1　线性规划的模型结构 ………………… 72
5.2　线性规划的单纯形法 ………………… 73
5.2.1　单纯形算法 ………………… 74
5.2.2　单纯形表格法的 MATLAB 程序：simplexTab ………… 75
5.3　linprog 函数 ………………… 80
5.3.1　实例演示1：（对应程序 test2） ………………… 82

5.3.2 实例演示 2：（对应程序 test4） …… 84
第 6 章 无约束优化算法 …… 86
6.1 最优性条件 …… 87
6.2 最速下降法 …… 88
6.2.1 算法原理 …… 88
6.2.2 算法步骤 …… 88
6.2.3 程序示例 …… 89
6.3 牛顿算法 …… 92
6.3.1 算法原理 …… 92
6.3.2 算法步骤 …… 93
6.3.3 算法特点 …… 93
6.4 拟牛顿算法（变尺度法） …… 94
6.4.1 算法原理 …… 94
6.4.2 算法步骤 …… 94
6.4.3 算法性质 …… 95
6.4.4 程序示例 …… 95
6.5 单纯形法 …… 98
6.5.1 算法原理 …… 99
6.5.2 函数 Fminsearch …… 100
6.6 含参数的优化问题 …… 101
6.7 大规模无约束优化问题 …… 103
第 7 章 约束优化算法 …… 105
7.1 罚函数法（内点法） …… 105
7.2 拉格朗日乘子法 …… 106
7.3 乘子法 MATLAB 程序及其使用 …… 107
7.3.1 Al_ main 函数 …… 107
7.3.2 乘子法 Al_ main 函数使用方法 …… 110
7.4 Fmincon 函数 …… 111
7.4.1 函数示例（1） …… 113
7.4.2 函数示例（2） …… 115
7.4.3 函数示例（3） …… 116
7.4.4 函数示例（4） …… 118
7.4.5 函数示例（5） …… 119
7.4.6 函数示例（6） …… 120
7.4.7 函数示例（7） …… 122
第 8 章 非线性最小二乘法 …… 124
8.1 高斯-牛顿法 …… 124
8.2 lsqnonneg 函数（求解非负约束的最小二乘问题） …… 126
8.3 lsqlin 函数（求解带约束的线性最小二乘问题） …… 128
8.3.1 函数示例（1） …… 129
8.3.2 函数示例（2） …… 131
8.4 lsqnonlin 函数（求解非线性最小二乘问题） …… 133
8.5 lsqcurvefit 函数（求解非线性数据拟合问题） …… 135
第 9 章 0-1 整数规划 …… 138
9.1 0-1 整数规划的基本模型 …… 138
9.2 分枝定界法与隐枚举法 …… 139
9.3 bintprog 函数（求解 0-1 整数规划） …… 141
9.3.1 函数示例（1） …… 142
9.3.2 函数示例（2） …… 144
9.4 分派问题 …… 145
9.4.1 指派问题的数学模型 …… 145
9.4.2 分派问题的转换及 AssignProb 函数 …… 146
9.4.3 AssignProb 函数示例（1） …… 148
9.4.4 AssignProb 函数示例（2） …… 150
9.4.5 AssignProb 函数示例（3） …… 151
第 10 章 目标规划 …… 154
10.1 目标规划模型 …… 154
10.1.1 问题提出 …… 154
10.1.2 目标规划模型的基本概念 …… 155
10.1.3 目标规划模型的一般形式 …… 157

10.1.4　利用 linprog 函数求解目标规划 …… 157
10.2　fgoalattain 函数 …… 159
10.2.1　函数示例（1） …… 161
10.2.2　函数示例（2） …… 162
第 11 章　最大最小问题 …… 164
11.1　最大最小问题模型 …… 164
11.2　fminimax 函数 …… 165
11.2.1　函数示例（1） …… 167
11.2.2　函数示例（2） …… 168
第 12 章　层次分析法（AHP） …… 172
12.1　层次分析法的基本概念 …… 172
12.1.1　建立系统的递阶层次模型 …… 172
12.1.2　构造判断矩阵 …… 174
12.1.3　单层权重计算 …… 174
12.1.4　各层元素对目标层的合成权重计算 …… 175
12.2　函数 AHPWeightVector（单层权重计算） …… 176
12.2.1　函数说明 …… 176
12.2.2　函数示例（1） …… 178
12.2.3　函数示例（2） …… 178
12.3　函数 AHPSolver（AHP 求解函数） …… 179
12.3.1　AHPSolver 代码 …… 179
12.3.2　AHPSolver 使用示例 …… 180
第 13 章　遗传算法 …… 185
13.1　遗传算法概要 …… 185
13.1.1　遗传算法模型 …… 185
13.1.2　遗传算法的特点 …… 186
13.1.3　遗传算法的发展 …… 187
13.1.4　遗传算法的应用 …… 188
13.1.5　基本遗传算法 …… 189
13.1.6　遗传算法的模式定理 …… 191
13.2　Genetic Algorithm Toolbox …… 195
13.2.1　函数概述 …… 195
13.2.2　函数使用说明及示例 …… 196
13.2.3　函数参数设置 …… 200
13.2.4　遗传算法 M 文件自动生成 …… 204
附录　MATLAB 优化工具箱参数设置 …… 206
参考文献 …… 216

第1章　运筹学概述

运筹学（Operations Research，OR）作为科学名词出现在20世纪30年代末。第二次世界大战期间，运筹学的研究与应用范围主要是战略、战术方面。随着世界性战争的结束，各国开始快速发展经济，世界范围内的剧烈竞争也体现在经济、技术方面，运筹学的研究也向这些方面拓展。运筹学为了适应时代的要求，在近几十年中，无论从理论上还是应用上都得到了快速的发展。在应用方面，今天运筹学已经涉及了服务、管理、规划、决策、组织、生产、建设等诸多方面，甚至可以说，很难找出它涉不到的领域。在理论方面，由于运筹学的需要和刺激而发展起来的一些数学分支，如数学规划，应用概率与统计、应用组合数学、对策论、数理经济学、系统科学等，都得到了迅速发展。

运筹学是一门应用科学，很难给出一个确切的定义。根据运筹学工作者的一些论述，我们可以较深切地理解这门科学的内涵。运筹学工作的先驱、诺贝尔奖金获得者、英国著名物理学家 P. M. S. Blackett 在1940年就开始从事运筹学方面的研究与应用。他曾多次指出：运筹学的一个明显的特征，正如目前所实践的，是它有或应该有一个严格且实际的性质，其目标是帮助人们找出一些方法，来改进正在进行中的或计划在未来进行的作战效率。为了达到这一目的，要研究过去的作战来明确事实，要得出一些理论来解释事实，最后利用这些事实和理论对未来的作战作出预测。我们可以逻列出一些论述：运筹学是“为决策机构在对其控制下业务活动进行决策时，提供以数量化为基础的科学方法。”“运筹学是一门应用科学，它广泛运用现有的科学技术知识和数学方法，解决实际中提出的专门问题，为决策者选择最优决策提供定量依据。”“运筹学是一种给出问题的坏答案的艺术，否则，问题的结果会更坏。”实质上，运筹学的基本目的是找到“优”的方案、途径，而在实际中，最优只能是一种理想追求。由于问题的复杂性，各种确定与不确定因素的综合影响，运筹学目标准确（或有保障的）定位，应该是通过研究使我们避开更坏的结果。从哲学角度讲，我们可以说运筹学就是用科学方法去了解和解释运行系统的现象。它在自然界范围内所选择的研究对象就是这些系统。这些系统时常包含着人和在自然环境中运行的“机器”，这个广义的“机器”可以推广为按照公认的规则运行的复杂社会结构。

我们称运筹学是一门科学，是因为它用科学方法来创建其知识。它与其他科学的不同之处在于，它研究的是运行系统的现象，这在其他科学中往往被忽略。

根据上述讨论，我们可以体会到，运筹学的思想就是对我们所关心的对象（“机器”或运行系统）进行深入的了解、分析和研究，特别是进行量化分析，得到信息，再合理运用有关的数学工具、系统方法等进行研究，提出有效方案来解决问题。

1.1 运筹学的特点及其应用

1.1.1 朴素运筹学思想及其深刻内涵

自从 1956 年引入以来，运筹学在我国已有四十多年的历史。经过这四十多年，运筹学在我国有了很大的发展，确立了它在经济建设中的地位。但是，运筹学在我国的发展状况与世界其他国家相比，尚有不小的差距，其中最主要的是认识与基础的问题。人们公认，将“Operations Research”译为“运筹学”最恰当。我国历史上，在军事和科学技术方面对运筹思想的运用是世界著名的，从春秋战国和三国时期的战争中就可举出很多运用运筹思想取得战争胜利的例子。这反映出运筹学注意系统数据采集、分析并研究优化方案的思想是一种朴素、自然的思想。实际上，很多人都在自觉不自觉地运用这种思想。另一方面，我们常说“道高一尺，魔高一丈”，在竞争中，各方共同运用这些思想解决问题时，就表现为对运筹学内涵研究、运用能力的提升。随着科学技术的发展，特别是信息社会的到来，运筹学内涵不断扩大，涉及的数学及其他基础科学的知识越来越多，于是熟练掌握并运用这门学科有效解决实际问题的难度也逐渐加大。根据运筹学的发展，数学、计算机科学及其他新兴学科的最新知识、技术都能很快融入其中，特别是人的直接参与决策，使得运筹学发展更进入一个崭新的阶段。这一切与我国科技发展的现状决定了我们必须加强对运筹学的研究与应用，只有这样才能逐步缩短与世界发达国家的距离。

1.1.2 运筹学研究的工作步骤

为了有效地应用运筹学，应当遵循下列六条原则：合伙原则、催化原则、独立原则、互相渗透原则、宽容原则和平衡原则。这些原则反映了运筹学工作者与其他各种因素的横向和纵向的联系。运筹学研究的工作步骤可以归纳为以下九个内容：

（1）目标的确定。即确定决策者期望从方案中得到什么。这个目标不应限制在过分狭小的范围内，但也要避免不必要的扩大；目标可以是一个，也可以是多个，随着和谐社会的发展，实际决策中都要考虑各个群体的利益，以至于多目

标规划目前在现实生活中越来越多地被使用了。

(2) 方案计划的研制。实施一项运筹学研究的过程常常是一个创造性的过程，计划的实质是规定出要完成某些子任务的时间，然后创造性地按时完成这一系列任务。这样做能够推动运筹学分析者得出结论，有助于方案的成功。对计划的任意延期和误时会导致分析者的消极工作和管理者的漠不关心。

(3) 问题的表述。这项工作的开展需要与管理人员进行深入讨论，经常包括与其他职员和业务人员的接触，采集必要的数据，以便了解问题的本质、历史及未来，以及问题各个变量之间的关系。这项任务的目的是为研究中的问题提供一个模型框架，并为以后的工作确立方向。在这里，第一要考虑问题是否能够分解为若干串行或并行的子问题；第二要确定模型建立的细节，如问题尺度的确定、可控制决策变量的确定、不可控制状态变量的确定、有效性度量的确定和各类参数、常数的确定。

(4) 模型的研制。模型是对各变量关系的描述，是成功解决问题的关键。构成模型的关系有几种类型，常用的有定义关系、经验关系和规范关系。

(5) 计算手段的拟定。在模型研制的同时，需要研究如何用数值方法求解模型，其中包括对问题变量性质（确定性、随机性、模糊性）、关系特征（线性、非线性）、手段（模拟、优化）及使用方法（现有的、新构造的）等的确定。例如，对于一个模型可能有许多算法可以适用，有的算法计算时间长但计算结果质量高，有的算法计算时间快但计算结果的质量相对较差，如何根据实际需要选择有效的算法，也是运筹学问题求解中重要的一环。

(6) 程序明细表的编制，程序设计和调试。对于计算过程需要编制程序来实现计算机运算的，运筹学研究应包含算法过程的描述、计算流程框图的绘制。程序的实现及调试可以交由程序员完成，或会同程序员完成。

(7) 数据收集。把有效性试验和实行方案所需的数据收集起来加以分析，研究输入的灵敏性，从而可以更准确地估计得到的结果。

(8) 方案验证。验证在运筹学的研究与应用中的重要性无论怎样强调都不会过分。验证包括两个方面：第一是确定验证模型，包括为验证一致性、灵敏性、似然性和工作能力而设计的分析与实验；第二是验证的进行，即把前一步收集到的数据用来对模型作完全试验。这样一种试验的结果，往往使模型必须重新设计，并要求与之相联系的程序重编。

(9) 方案实施。运筹学分析者往往认为，在模型验证后，任务就完成了，这是不对的。事实上，一项研究的真正困难往往在方案的最后一步，即在实施和维护时才暴露出来。因此要使得整个研究有效，必须和那些与所研究的决策问题相关的各职能的各级管理人员进行合作，并让他们参与其中。

1.2　运筹学建模

1.2.1　运筹学建模的一般思路

运筹学建模在理论上应属于数学建模的一个部分。因此，运筹学建模所采用的手段、途径就是数学建模中所采用的。本节所要介绍的，是根据运筹学本身的特征来处理建模问题的一般思路。

经过长期、深入的研究和发展，把运筹学处理的问题归纳成一系列具有较强背景和规范特征的典型问题。因此，运筹学建模就要把相当的精力放在将实际问题合理地描述为某种典型的运筹模型。在这个过程中，要求运筹学工作者具有以下几个方面的知识和能力：

（1）熟悉典型运筹模型的特征和它的应用背景，如线性规划、非线性规划、整数规划、层次分析法（AHP）等。

（2）有理解实际问题的能力，包括广博的知识、搜集信息、资料和数据的能力。

（3）有抽象分析问题的能力，包括抓主要矛盾、逻辑思维、推理、归纳、联想、类比等创造能力。

（4）有运用各类工具知识的能力，包括运用数学知识、计算机、自然科学和工程技术等的能力。

（5）有试验校正、维护修正模型的能力。

根据问题本身的情况，按照上节的讨论，我们在建模时一般有如下思路：

（1）直接方法。当我们熟悉问题的内在关系、特征以及运筹学的典型模型特点时，常常可以直接得到一些问题的模型或问题归类，即确定问题是属于线性规划、非线性规划、整数规划、排队模型等的哪一种。有时模型的参数也可直接从问题本身得到。

（2）类比方法。通过类比把新遇到的问题用已知类似问题的模型来建立该问题的模型。这种情况往往得到的是模型归类，而模型参数需用其他方法取得。

（3）模拟方法。利用计算机程序实现对问题的实际运行模拟，可得到有用的数据。这些数据常用来求得模型参数或对所建立模型的合理性、正确性的检验。

（4）数据分析法。利用数据处理的方法分析各数据变量之间的关系是确定关系还是相关关系，以及是何种相关等。这种方法还可以用回归分析找出变量的变化趋势，从而得到合理的数学模型。大量的模型参数求得也常常使用数据处理

的统计方法。另外，回归模型常常就是一个无约束最优化模型。

（5）试验分析法。通过试验分析建模是工程管理中常用的方法。以局部的试验产生数据，经过统计处理得到总体的模型或模型归类。试验分析更多地用于产生模型参数。

1.2.2　运筹学模型的评价

一个好的运筹学模型，不仅能比较真实地反映实际问题，还要具备以下几个优点：

（1）易于理解。模型应力求简明。这里要强调一点，模型越大越复杂，不一定意味着越好。应当把实际问题中那些不重要的因素删去。这样，一方面，形成模型以后，由于变量和约束个数较少，便于计算求解；另一方面，也更易于揭示主要因素对问题的影响以及它们之间的关系。

模型中的变元、函数符号要接近所代表的实际因素、资源和目标的原义，这样易于理解模型所表示的实际问题的结构，而且当变量和约束个数很多时，对于模型建立和解释也是便利的。

（2）易于探查错误。如果上面一点做得比较好，那么模型也易于探查错误。模型的错误一般有两种：①书写错误；②模型与实际问题不符。后一种错误在建立模型时应尽量避免，在评价模型及其解时，也可以找出错误并改正。前一种错误的避免，一方面要求细心，另一方面要求模型的书写形式要规范，变量次序最好固定不变。如果在某个函数关系中不出现某个变量，那么该变量的位置最好以空白代替；在约束条件中，把等式约束与不等式约束分成两组，分开来写，而在不等式约束中又可以把“≥”不等式与“≤”不等式分开来写。为便于清楚地表明每个约束所代表的对实际资源的限制，不必把两种形式的不等式统一成一种。

（3）易于计算。运筹学模型问题是否易于求解，取决于问题的规模、复杂程度、当前的计算技术水平和解该问题的算法。可以通过以下几个途径降低问题的复杂程度和规模：

1）采用简单的函数。与其在建模时花费许多时间、精力，寻找很好地反映现实情况但非常复杂的函数，不如在误差允许的范围内，采用简单的函数。在这一点上，尤其应该注意的是，非线性函数在计算方面的困难远比线性函数大得多。

2）删去不必要的变量和约束条件。在建立模型前，分析问题时就应该注意把那些不重要的因素和资源限制简化掉。建立模型后，则应注意删去那些多余的约束条件。多余的约束条件是指该约束条件被删去后，可行解集没有改变。一个

简单的例子是：$t<1$ 和 $t<6$，这里 $6>1$，那么条件 $t<6$ 就是多余的。尤其要删去那些非线性的约束条件。但是没有实用的一般方法能识别模型中的多余约束条件。尽管如此，在求解运筹学模型前，对它进行一番数学上的分析、讨论它的性质仍是非常必要的。

3）函数变换。通过将复杂的函数进行代数处理，可以达到降低模型复杂程度的目的。

例如：

(1) $\text{s.t.}\begin{cases}(x_1+x_2)^2-(x_1-x_2)^2\leqslant 10\\ x_1,\ x_2\leqslant 0\end{cases}\Rightarrow \text{s.t.}\begin{cases}4x_1x_2\leqslant 10\\ x_1,\ x_2\leqslant 0\end{cases}$

(2) $\prod_n x_n^{a_n}\leqslant c\Rightarrow\sum_n a_n\ln x_n\leqslant \ln c$

上面两个例子分别根据因式分解方法和连乘函数转为对数的连加函数的方法，从计算的角度对约束函数进行了简化。

1.2.3 运筹学模型的求解

解决一类优化问题，一般会有多种算法可供选择，某些特别的问题也有专门的算法处理。某个算法对某类问题特别有效，但对于其他问题也许根本不起作用。建立模型时，要注意到哪些算法对求解该类问题是有效的，在计算误差与计算时间允许的范围内，选择那些相对比较有效的算法去求解该问题。因此，模型建立者应对优化算法进行系统的了解，熟悉每种算法的优势与缺点。例如以下两类问题对算法的速度要求较高：

例 1-1：投资组合优化配置就是优化算法在金融中的主要应用。由于市场的行情时刻在变化，原有投资组合在当前的市场情况中是否最优，直接决定着该组合的收益。对于投资组合包含投资品种较多的时候（例如 J. P. 摩根管理着四千多种资产），优化问题的计算量巨大，在有限的时间内得到全局最优似乎不可能，而且市场的信息多数也是不精确的，对于这些不精确的参数即使求出全局最优解，解的实际精度也会遭到质疑。如果将投资组合优化系统做成实时系统似乎也不太现实，一般只要求在原有投资组合的配置下根据市场变化有所改进就可接受。

例 1-2：炼油设备中的压力控制系统在相关参数基本确定的情况下，管道内压力的大小决定着炼油效率的高低，但压力的变化又同时影响着其他参数。这样的一个压力优化控制系统必须是实时的，但该优化问题的目标函数中涉及了复杂的流体力学的偏微分方程求解问题。这类问题不可能在短时间内求得最优解，只能在规定的反应时间内求得较有效解。

在实践中，最优化算法是求解运筹学问题时使用的主要算法。根据优化算法理论发展与算法原型将现有的最优化算法分为：经典优化算法和启发式优化算法两大类。经典优化算法和启发式优化算法都是迭代算法，但是，它们又有很大区别，如：

（1）经典算法以一个可行解为迭代的初始值，而启发式算法以一组可行解为初始值。

（2）经典算法的搜索策略为确定型的，而启发式算法的搜索策略是结构化和随机化的。

（3）经典算法大多都需要导数信息，而启发式算法仅用到目标函数值的信息。

（4）经典算法对函数性质有着严格要求，而启发式算法对函数性质没有太大要求。

（5）经典算法的计算量要比启发式算法小很多，比如，对于规模较大且函数性质比较差的优化问题，经典算法的效果不好，但一般的启发式算法的计算量太大。

在计算时，优化算法的迭代过程主要由找寻搜索方向和确定搜索步长组成。搜索方向和搜索步长的选取决定了优化算法的搜索广度和搜索深度。经典优化算法和启发式优化算法的区别主要是其搜索机制不同。经典算法的搜索方向和搜索步长是由局部信息（如导数）决定的，所以一般只能对局部进行有效的深度搜索，而不能进行有效的广度搜索，所以经典优化算法很难跳出局部最优；启发式优化算法为了避免像经典优化算法那样陷入局部最优，采用了相对有效的广度搜索，但是在问题规模较大的时候，这样做的结果往往使得计算量难以承受。

纵观优化算法的发展，完美的算法是不存在的。常用来评价算法好坏的标准有：

（1）算法收敛速度。

（2）算法使用范围（普适性）。

（3）算法的时间复杂度。

（4）算法得到解的质量（局部性或全局性，对绝对最优解的近似程度）。

（5）算法的可实现性。

可以说这些标准是不可公度的（不可能同时都好）。以全局最优问题为例，如果要求计算时间少，那么搜索广度就无法保证，解的质量就差；如果要求收敛速度快，就需要有效的搜索方向，有了搜索方向就降低了搜索广度，这样解的全局最优性无法保证。

计算复杂性理论，全局优化问题是 NP-complete 问题，一般根据实际问题采

用不同的启发式算法。算法的评价标准不可公度，而且在具体问题中，这些标准也不是等重要的。比如某些问题对解的要求降低 10%，它的计算时间就可以减少 50%，这样做是否值得，要根据实际情况而定。

1.3 基本概念和符号

1.3.1 空间与向量

本书的算法理论是建立在 n 维欧氏空间基础上的，书中使用下列符号：

$\mathbf{R}^n$ 表示 n 维欧氏空间：$\boldsymbol{x}\in\mathbf{R}^n$，表示 $\boldsymbol{x}$ 为 $\mathbf{R}^n$ 中的向量，若不进行特殊说明，本书中 $\boldsymbol{x}=(x_1, x_2, \cdots, x_n)^{\mathrm{T}}$，其中 x_i 为 $\boldsymbol{x}$ 的第 i 维分量，符号“T”表示转置。

设 $\boldsymbol{x}, \boldsymbol{y}\in\mathbf{R}^n$，$\boldsymbol{x}, \boldsymbol{y}$ 的内积记为 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}=\sum_{i=1}^{n}x_iy_i$，在不加说明时，我们用 $\|\cdot\|$ 表示 2 范数，$\|\boldsymbol{x}\|=\sqrt{\sum_{i=1}^{n}x_i^2}$。

1.3.2 梯度向量与 Hesse 矩阵

设 $f(\boldsymbol{x}):\mathbf{R}^n\to\mathbf{R}$，$f$ 二次可微。

定义：设 $f(\boldsymbol{x})$ 在 $S\subset\mathbf{R}^n$ 上有定义，$\boldsymbol{x}^{(0)}\in\mathrm{int}S$，若 $\exists p\in\mathbf{R}^n$，使得 $\forall\boldsymbol{x}\in S$，有：

$$f(\boldsymbol{x})=f(\boldsymbol{x}^{(0)})+\boldsymbol{p}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}^{(0)})+\|\boldsymbol{x}-\boldsymbol{x}^{(0)}\|\alpha(\boldsymbol{x}-\boldsymbol{x}^{(0)}) \tag{1-1}$$

其中，$\lim_{\boldsymbol{x}\to\boldsymbol{x}^{(0)}}\alpha(\boldsymbol{x}-\boldsymbol{x}^{(0)})=0$，则称 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}^{(0)}$ 处可微，向量 $\boldsymbol{p}$ 为 f 在 $\boldsymbol{x}^{(0)}$ 处的梯度，记为：$\nabla f(\boldsymbol{x}^{(0)})=\boldsymbol{p}$。

如果进一步存在对称的 n 阶方阵 $\boldsymbol{H}$，使得 $\forall\boldsymbol{x}\in S$，有：

$$f(\boldsymbol{x})=f(\boldsymbol{x}^{(0)})+p^{\mathrm{T}}(\dot{\boldsymbol{x}}-\boldsymbol{x}^{(0)})+\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^{(0)})^{\mathrm{T}}H(\boldsymbol{x}-\boldsymbol{x}^{(0)})+\|\boldsymbol{x}-\boldsymbol{x}^{(0)}\|^2\beta(\boldsymbol{x},\boldsymbol{x}^{(0)}) \tag{1-2}$$

其中，$\lim_{\boldsymbol{x}\to\boldsymbol{x}^{(0)}}\beta(\boldsymbol{x}-\boldsymbol{x}^{(0)})=0$，则称 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}^{(0)}$ 处二阶可微，方阵 $\boldsymbol{H}$ 成为 f 在 $\boldsymbol{x}^{(0)}$ 处的 Hesse 矩阵，记为：$\nabla^2 f(\boldsymbol{x}^{(0)})=\boldsymbol{H}$。

不难得到，当 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}^{(0)}$ 处二阶可微时：

$$\nabla f(\boldsymbol{x}^{(0)})=\left(\frac{\partial f(\boldsymbol{x}^{(0)})}{\partial x_1},\frac{\partial f(\boldsymbol{x}^{(0)})}{\partial x_2},\cdots,\frac{\partial f(\boldsymbol{x}^{(0)})}{\partial x_n}\right)^{\mathrm{T}}$$

$$\nabla^2 f(\boldsymbol{x}^{(0)}) = \begin{pmatrix} \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_n \partial x_1} \\ \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_n \partial x_2} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_1 \partial x_n} & \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f(\boldsymbol{x}^{(0)})}{\partial x_n^2} \end{pmatrix} \tag{1-3}$$

$f(\boldsymbol{x})$在$\boldsymbol{x}^{(0)}$处的一阶泰勒展开式为：

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^{(0)}) + \nabla f(\boldsymbol{x}^{(0)})^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^{(0)}) + o(\|\boldsymbol{x} - \boldsymbol{x}^{(0)}\|) \tag{1-4}$$

$f(\boldsymbol{x})$在$\boldsymbol{x}^{(0)}$处的二阶泰勒展开式为：

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^{(0)}) + \nabla f(\boldsymbol{x}^{(0)})^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^{\mathrm{T}} \nabla^2 f(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}) + o(\|\boldsymbol{x} - \boldsymbol{x}^{(0)}\|^2) \tag{1-5}$$

1.3.3　点和方向

在解数学规划问题时，常常要涉及迭代点和搜索方向，点和方向在 R^n 中都是向量，我们涉及的点是 $\mathbf{R}^n$ 中的一个固定的向量，一般用$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\cdots$来标记；本书中谈到的方向，是指非零自由向量，它有固定的长度和方向，但起点可以移动，一般用$\boldsymbol{d},\boldsymbol{s},\cdots$来标记。

图　1-1

如$\boldsymbol{x}+\lambda\boldsymbol{d}$，表示从$\boldsymbol{x}$点出发沿方向$\boldsymbol{d}$，移动$\boldsymbol{d}$长度的$\lambda$倍所得到的向量，如图 1-1 所示。函数$f(\boldsymbol{x})$，表示沿这个方向的斜率为$f_\lambda'(\boldsymbol{x}+\lambda\boldsymbol{d})|_{\lambda=0} = \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$，称关于$\lambda$的二阶导数为函数$f(\boldsymbol{x})$沿此方向的曲率。

$$f_\lambda''(\boldsymbol{x}+\lambda\boldsymbol{d})|_{\lambda=0} = \boldsymbol{d}^{\mathrm{T}} \nabla^2 f(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{d}$$

定义：$S \subset \mathbf{R}^n$，非空，$f:S \to \mathbf{R}$，$\boldsymbol{x} \in S$，$\boldsymbol{d} \in \mathbf{R}^n$，$\boldsymbol{d} \neq 0$，使当$\lambda > 0$充分小时有$\boldsymbol{x}+\lambda\boldsymbol{d} \in S$，如果极限$\lim\limits_{\lambda \to 0^+}\frac{f(\boldsymbol{x}+\lambda\boldsymbol{d})-f(\boldsymbol{x})}{\lambda}$存在（包括取$\pm\infty$），则称$f$在$\boldsymbol{x}$点沿$\boldsymbol{d}$方向有导数，记为：

$$f'(\boldsymbol{x};\boldsymbol{d}) = \lim_{\lambda \to 0^+}\frac{f(\boldsymbol{x}+\lambda\boldsymbol{d})-f(\boldsymbol{x})}{\lambda}$$

显然，当f在$\boldsymbol{x}$点可微时，$f'(\boldsymbol{x};\boldsymbol{d}) = \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$。

第2章　基本概念和基本理论

凸分析是运筹学的支柱性理论，优化理论大多建立在凸集合与凸规划的基础之上。进而，根据优化问题可行解集合、目标与约束函数的凸性，将优化问题的解分为局部最优解与全局最优解。目前，优化算法种类繁多，根据其优化理论，将现有优化算法分为经典优化算法与启发式优化算法。运筹学与最优化的发展与其他科学理论的发展密切相关，如数值计算、计算复杂性等。

本章将主要对凸集合与凸规划、局部最优解与全局最优解、经典优化算法与启发式优化算法、数值计算与计算复杂性等作综述性的简单介绍。

2.1　基本概念

在现实生活中，许多重要的问题都涉及选取一个最好的目标，或者为达到这个目标而选择某些参数、确定某些值，这些问题都可以归结为最优化问题。对于一个最小值问题，其数学规划模型的一般形式为：

$$(fS)\ \begin{cases}\min f(\boldsymbol{x}) \\ \text{s.t. } \boldsymbol{x}\in S\end{cases} \tag{2-1}$$

其中，$S\subset \mathbf{R}^n$ 为约束集合或可行集；f: $S\rightarrow \mathbf{R}$ 为目标函数；若 $\boldsymbol{x}\in S$，则称 $\boldsymbol{x}$ 为问题（fS）的可行解。显然，只要改变目标函数的符号，最大值问题就可以转变成最小值问题。因此，本书一般以最小值问题为标准。解决最优化问题的算法称为最优化算法，可以分为经典优化算法和启发式优化算法。

若 $\boldsymbol{x}^*\in S$，且满足 $\forall \boldsymbol{x}\in S$，$f(\boldsymbol{x}^*)\leqslant f(\boldsymbol{x})$，则称 $\boldsymbol{x}^*$ 是问题（fS）的最优解，记为 opt.。对于 $f^*=f(\boldsymbol{x}^*)$，称之为问题（fs）的最优值。

定义 2-1　考虑问题（fs），设 $\boldsymbol{x}^*\in S$，有：

（1）若 $\forall \boldsymbol{x}\in S$，恒有 $f(\boldsymbol{x}^*)\leqslant f(\boldsymbol{x})$，则称 $\boldsymbol{x}^*$ 是问题（fs）的全局最优解，记为 g. opt.（global optimum）或 opt.。若当 $\boldsymbol{x}\neq \boldsymbol{x}^*$ 时，严格不等式 $f(\boldsymbol{x}^*)<f(\boldsymbol{x})$ 成立，则称 $\boldsymbol{x}^*$ 是问题（fS）的严格全局最优解。

（2）若存在 $\boldsymbol{x}^*$ 的邻域 $N(\boldsymbol{x}^*)$，使得 $\forall \boldsymbol{x}\in S\cap N(\boldsymbol{x}^*)$ 恒有 $f(\boldsymbol{x}^*)\leqslant f(\boldsymbol{x})$，则称 $\boldsymbol{x}^*$ 是问题（fS）的局部最优解，记为 l. opt.（local optimum）；若当 $\boldsymbol{x}\neq \boldsymbol{x}^*$ 时，严格不等式 $f(\boldsymbol{x}^*)<f(\boldsymbol{x})$ 成立，则称 $\boldsymbol{x}^*$ 是问题（fs）的严格局部最优解。

定义 2-2　设 $S\subset \mathbf{R}^n$，若对于任意 $\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)}\in S,\lambda\in[0,1]$，均有 $\lambda\boldsymbol{x}^{(1)}+(1-$

$\lambda)x^{(2)} \in S$,则称集合 S 为凸集。

由定义易知,凸集的几何意义是,对于非空集合 $S \subset \mathbf{R}^n$,连接 S 中任意两点 $x^{(1)}, x^{(2)}$ 的线段仍属于该集合。

为了便于归类和讨论,规定空集 $\varnothing$ 为凸集,单点集 $\{x\}$ 为凸集。

定义 2-3　设 $x^{(1)}, x^{(2)}, \cdots, x^{(m)} \in \mathbf{R}^n, \lambda_i \geqslant 0, i=1,2,\cdots,m$,且 $\sum_{i=1}^{m} \lambda_i = 1$,则称 $\sum_{i=1}^{m} \lambda_i x^{(i)}$ 为点 $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$ 的凸组合。

设 $S \subset \mathbf{R}^n$,由 S 中所有有限点的凸组合构成的集合为凸集,称为 S 的凸包,记作 $\mathrm{cov}(S)$。显然,若 S 为凸集,则 $S = \mathrm{cov}(S)$。

2.2　经典优化算法

2.2.1　线性最优化

线性最优化又称线性规划，是运筹学中应用最广泛的一个分支。这是因为自然科学和社会科学中许多问题都可以近似转化成线性规划问题。

线形规划的一般形式为：

$$
\begin{aligned}
&\min z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
&\text{s.t.}\ \ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leqslant b_1 \\
&\qquad a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leqslant b_2 \\
&\qquad \vdots \\
&\qquad a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leqslant b_m \\
&\qquad x_1,\ x_2,\ \cdots,\ x_n \geqslant 0
\end{aligned}
\tag{2-2}
$$

线性规划理论和算法的研究及发展共经历了三个发展阶段，每个阶段都引起了社会的极大关注。线性规划研究的第一高潮是著名的单纯形法的研究。这一方法是 Dantzig 在 1947 年提出的，它以成熟的算法理论和完善的算法及软件统治线性规划达三十多年。随着 20 世纪 60 年代发展起来的计算复杂性理论的研究，单纯形法在 70 年代末受到了挑战。1979 年，前苏联数学家 Khachiyan 提出了第一个理论上优于单纯形法的所谓多项式时间算法——椭球法，曾成为轰动一时的新闻，并掀起了研究线性规划的第二个高潮。但遗憾的是，广泛的数值试验表明，椭球法的计算比单纯形方法差。1984 年，Karmarkar 提出了求解线性规划的另一个多项式时间算法。这个算法从理论和数值上都优于椭球法，因而引起了学术界的极大关注，并由此掀起了研究线性规划的第三个高潮。从那以后，许多学者致

力于改进和完善这一算法，得到了许多改进算法。这些算法运用不同的思想方法均获得通过可行区域内部的迭代点列，因此统称为解线性规划问题的内点算法。目前，面对大规模的线性规划问题，内点算法将以不可抗拒的趋势超越和替代单纯形法。

目前成熟的解线性和整数规划问题的软件有：Lindo、EQPS（线性、整数和非线性规划）、FMP（线性和混合整数规划）、HS/LPLO（线性规划）、KORBX（线性规划）、LAMPS（线性和整数规划）、LPBLP（线性规划）、MILP（混合整数规划）、MINTO（混合整数规划）、MPSIII（线性和混合整数规划）、OML（线性和混合整数规划）、OSL（线性、二次和混合整数规划）、PROCLP（线性和整数规划）、WB（线性和混合整数规划）、WHIZARD（线性和混合整数规划）、XPRESSMP（线性和混合整数规划）等。

2.2.2 非线性最优化

非线形规划的一般形式为：

$$\begin{aligned}&\min f(x)\\&\text{s.t. } g_i(x)\leqslant 0 \quad i=1,2,\cdots,m\\&\qquad h_j(x)=0 \quad j=1,2,\cdots,l\end{aligned} \tag{2-3}$$

$f,g_i,h_j:\mathbf{R}^n\to\mathbf{R}$，其中至少有一个为非线性函数

有 m 个不等式约束条件

有 l 个等式约束条件

非线性规划的一个重要理论是 1951 年提出的 Kuhn-Tucker 最优条件（简称 K-T 条件），此后的 50 年代主要是对梯度法和牛顿法的研究。以 Davidon（1959），Fletcher 和 Powell（1963）提出的 DFP 方法为起点，60～80 年代是研究拟牛顿方法的活跃时期，同时对共轭梯度法也有较好的研究。在 1970 年，由 Broyden，Fletcher，Goldfarb 和 Shanno 从不同的角度共同提出的 BFGS 方法是目前为止最有效的拟牛顿方法。由于 Broyden，Dennis 和 More 的工作使得拟牛顿方法的理论变得很完善，70 年代是非线性规划的飞速发展时期，约束变尺度（SQP）方法（Han 和 Powell 为代表）和 Lagrange 乘子法（代表人物是 Powell 和 Hestenes）是这一时期的主要研究成果。计算机的飞速发展使非线性规划的研究如虎添翼，80 年代开始研究信赖域法、稀疏拟牛顿法、大规模问题的方法和并行计算，90 年代研究解非线性规划问题的内点法和有限储存法。这半个世纪是最优化发展的黄金时期。

与线性规划相比，非线性规划软件还不够完善。但是已有大量求解非线性规划问题的软件，其中有相当一部分可从互联网上免费下载。LANCELOT 是由

Conn，Gould 和 Toint 研制的解大规模最优化问题的软件包，适合求解无约束最优化、非线性最小二乘、边界约束最优化和一般约束最优化问题。这个软件的基本思想是利用增广 Lagrange 函数来处理约束条件，在每步迭代中解一个边界约束优化子问题，其所用的方法结合了信赖域和投影梯度等技术。MINPACK 是美国 Argonne 国家实验室研制的软件包，适合求解非线性方程组和非线性最小二乘问题，所用的基本方法是阻尼最小二乘法，此软件可以从网上图书馆获得。PROC-NLP 是 SAS 软件公司研制的 SAS 商业软件中 OR 模块的一个程序，这个程序适合求解无约束最优化、非线性最小二乘、线性约束最优化、二次规划和一般约束最优化问题。TENMIN 是 Schnabel 等研制的解中小规模问题的张量方法软件。目前成熟的解非线性最优化问题的软件有：Lingo、CONOPT（非线性规划）、DOT（优化设计工具箱）、Excel and Quattro Pro Solvers（线性、整数和非线性规划）、FSQP（非线性规划和极小极大问题）、GRG2（非线性规划）、LBFGS（有限储存法）、LINDO（线性、二次和混合整数规划）、LSSOL（最小二乘和二次规划）、MINOS（线性和非线性规划）、NLPJOB（非线性多目标规划）、OPTPACK（约束和无约束最优化）、PETS（解非线性方程组和无约束问题的并行算法）、QPOPT（线性和二次规划）、SQOPT（大规模线性和凸二次规划）、SNOPT（大规模线性、二次和非线性规划）、SPRNLP（稀疏最小二乘，稀疏和稠密非线性规划）、SYSFIT（非线性方程组的参数估计）、TENSOLVE（非线性方程组和最小二乘）、VE10（非线性最小二乘）等。

2.3　启发式算法

大自然是神奇的，它造就了很多巧妙的手段和运行机制。受大自然的启发，人们从大自然的运行规律中找到了许多解决实际问题的方法。对于那些由大自然的运行规律或者面向具体问题的经验、规则启发出来的方法，人们常常称之为启发式算法（Heuristic Algorithm）。现在的启发式算法也不是全部来自然的规律，也有来自人类积累的工作经验。

启发式算法有不同的定义：一种定义为，一个基于直观或经验构造的算法，对优化问题的实例能在可接受的计算成本（计算时间、占用空间等）内，给出一个近似最优解，该近似最优解与真实最优解的偏离程度不一定可以事先预计；另一种是，启发式算法是一种技术，这种技术使得在可接受的计算成本内去搜寻最好的解，但不一定能保证所得的是可行解和最优解，甚至在多数情况下，无法阐述所得解同最优解的近似程度。本文较倾向赞同第二种定义，因为启发式算法现在还没有完备的理论体系，只能视作一种技术。由于启发式算法的计算量都比

较大，所以伴随着计算机技术的发展，启发式算法取得了巨大的成就。

20 世纪 40 年代：由于实际需要，人们已经提出了一些解决实际问题的快速有效的启发式算法。

20 世纪 50 年代：启发式算法的研究逐步繁荣起来。随后，人们把启发式算法的思想和人工智能领域中的各种有关问题的求解的收缩方法结合起来，提出了许多启发式的搜索算法。其中贪婪算法和局部搜索算法等受到人们的关注。

20 世纪 60 年代：随着人们对数学模型和优化算法的研究越来越重视，发现以前提出的启发式算法速度很快，但是解的质量不能保证。虽然对优化算法的研究取得了很大的进展，但是对较大规模的问题仍然无能为力（计算量还是太大）。

20 世纪 70 年代：计算复杂性理论的提出。NP 完全理论告诉我们，许多实际问题不可能在合理的时间范围内找到全局最优解。贪婪算法和局部搜索算法速度快，但解不好的原因主要是它们只是在局部区域内找解，得到的解不能保证全局最优。因此必须引入新的搜索机制和策略，才能有效地解决这些困难问题，这就导致了超启发式算法（Meta-heuristic Algorithms）的产生。Holland 模拟地球上生物进化规律提出了遗传算法（Genetic Algorithm），它与众不同的搜索机制再次引发了人们研究启发式算法的兴趣，从而掀起了研究启发式算法的热潮。

20 世纪 80 年代以后：模拟退火算法（Simulated Annealing Algorithm）、人工神经网络（Artificial Neural Network）、禁忌搜索（Tabu Search）相继出现。最近，演化算法（Evolutionary Algorithm）、蚁群算法（Ant Algorithms）、拟人拟物算法、量子算法等又相继兴起，掀起了研究启发式算法的高潮。由于这些算法简单和有效，而且具有某种智能，因而成为科学计算和人类实践之间的桥梁。

优胜劣汰是大自然的普遍规律，主要通过选择和变异来实现。选择是优化的基本思想，变异（多样化）是随机搜索或非确定搜索的基本思想。“优胜劣汰”是算法搜索的核心，根据“优胜劣汰”策略的不同，可以获得不同的超启发式算法。超启发式算法的主要思想来自于人类经过长期对物理、生物、社会等自然现象仔细的观察和实践，以及对这些自然现象的深刻理解，逐步向大自然学习，模仿其中的自然现象的运行机制而得到的。

（1）遗传算法：是根据生物演化，模拟演化过程中基因染色体的选择、交叉和变异得到的算法。在进化过程中，较好的个体有较大的生存几率。

（2）模拟退火：模拟统计物理中固体物质的结晶过程。在退火过程中，如果搜索到好的解就接受；否则，以一定的概率接受不好的解（即实现多样化或变异的思想），达到跳出局部最优解的目的。

（3）神经网络：模拟大脑神经处理事务，通过各个神经元的竞争和协作，

实现选择和变异的过程。

（4）禁忌搜索：模拟人的经验，通过禁忌表记忆最近搜索过程中的历史信息，禁忌某些解，以避免走回头路，达到跳出局部最优解的目的。

（5）蚂蚁算法：模拟蚂蚁的行为，向蚂蚁的协作方式学习。

以上几种超启发式算法都有一个共同的特点：从随机的可行初始解出发，采用迭代改进的策略去逼近问题的最优解。

它们的基本要素包括：

（1）随机初始可行解。

（2）给定一个评价函数（主要与目标函数值有关）。

（3）产生新的可行解。

（4）选择和接受解的准则。

（5）终止准则。

其中（4）集中反映了超启发式算法的克服局部最优的能力。

虽然人们对启发式算法的研究了将近 50 年，但它还有很多方面的研究存在不足，如：

（1）启发式算法目前缺乏统一、完整的理论体系。

（2）由于 NP 理论，各种启发式算法都不可避免地遭遇到局部最优的问题，即如何判断最优性。

（3）启发式算法都有各自优点，但缺乏关于各算法完美结合的研究。

（4）启发式算法中的参数对算法的效果起着至关重要的作用，关于有效设置参数方面缺乏深入研究。

（5）启发式算法缺乏有效的迭代停止条件。

（6）关于启发式算法收敛速度的研究尚不充分。

2.4　全局最优与计算复杂性

全局最优化的问题很早就有人提出来了。Markowzi，Manne（1957）和 Dantzig（1958）讨论了线性混合整数规划问题的全局最优解（线性优化问题没有全局最优和局部最优的区别，因为线性规划是凸规划，它的可行域上的局部最优就是全局最优），之后又有 LAND 和 DOIG 等研究了全局最优化问题。全局最优化成为数学规划的一个重要分支的时间是 19 世纪 70 年代后期（“Towards Global Optimization”，dixon，Szego，1975）。为了解决这个问题，许多学者开始研究它，并提出了丰富的理论和算法成果，但是人们发现似乎是没有好的算法能有效地解决全局最优化问题。

起初人们用经典算法（如最速下降法、牛顿法、SQP 等）进行多初始点的计算，但不能有效地解决全局最优化问题。与此同时，计算复杂性理论（19 世纪 70 年代）创立了。自从 Cook1972 年提出 NP 理论后，衡量计算复杂性有了标准，人们进而证明了全局最优化问题是 NP 问题。经典算法求解全局最优化问题的效果不是很好，随后有的人开始根据全局优化问题的特点对经典算法进行了改进，有的则引入了启发式算法（如遗传算法、神经网络、模拟退火等），大都采用随机搜索的策略。完美的东西似乎是不存在的，如果得到精确的解，计算时间总是难以承受。计算时间的减少和计算结果质量的提高是不能同时做到的，如果想提高计算结果的质量就要多花时间，如果想要快速计算，计算结果的最优性就无法保证。

有时各种算法对同一个问题都有可能给出最优解，为了判定各种算法的效率，人们给出了算法复杂性的度量，以及几个多项式和指数的时间复杂性函数的对比表，如表 2-1 所示。计算复杂性理论是研究算法有效性和问题难度的一种工具，它是最优化问题的基础，涉及如何判断一个问题的难易程度。只有了解了所研究问题的复杂性，才能更有针对性地设计相关算法，以及提高算法效率。

所谓 P 问题，就是可以在关于问题本身参数（如维数、约束个数等）的多项式时间内求解的问题。

表 2-1　几个多项式和指数的时间复杂性函数的对比表

时间复杂性函数	规模 n					
	10	20	30	40	50	60
N	0.00001s	0.00002s	0.00003s	0.00004s	0.00005s	0.00006s
N^2	0.0001s	0.0004s	0.0009s	0.0016s	0.0025s	0.0036s
N^3	0.001s	0.008s	0.027s	0.064s	0.125s	0.216s
N^5	0.1s	3.2s	24.3s	1.7m	5.2m	13.0m
2^n	0.001s	1.0s	17.9m	12.7d	35.7y	366c
3^n	0.059s	58m	6.5y	3855c	2×10^8c	1.3×10^{13}c

注：表中，s：秒；m：分；d：天；y：年；c：世纪。

NP 问题就是对于一个给定的点，能在多项式时间内判定它是否给定问题的解的问题。NP 包含 P 是显然的事实。但是 P 是否也包含 NP，就是一个非常困难的问题。目前，这个问题被列为世界 7 大数学难题之首。有一类 NP 问题，它们之间相互等价，求解其中一个问题就求解了全部问题，大部分组合优化问题属于此类。这类问题称为 NP 完全问题类，单个问题就称为 NP 完全问题。共识性的看法认为，这类问题不存在多项式时间算法。

2.5　计算误差理论

在数值计算过程中，往往会出现各种各样的误差，它们会直接影响到计算的结果，这些误差也会在计算过程中产生各种各样的效益。例如，某个参数由于观测引起的误差是微不足道的，或者少量的舍人误差对中间计算的结果影响不是很大，但是，这些误差经过计算机的千万次运算以后，误差的积累可能大得惊人。初始数据的微小误差也可能会引起严重错误，甚至会导致完全错误的结果。因此，挑选和设计好的算法，是一个很重要的环节，必须加以足够的重视。

2.5.1　误差产生的原因和形式

误差在数值计算中是不可避免的。误差主要有截断误差（方法误差）和舍人误差两种。

1. 截断误差

算法的实现是从连续系统到离散系统的一个转变过程。许多数学运算（如微分、积分与无穷级数求和等）是通过极限过程来定义的，而实际上计算机只能完成有限次的算术与逻辑运算。因此，在实际计算过程中，还需将解题方案加工成算术运算和符号运算的有限序列，而这种加工往往又表现为某无限过程的一个“截断”。例如，对于收敛的无穷级数，常用它前面的有限项的和代替无穷级数的和，实际上抛弃了无穷级数的后段，由此便产生了误差。

2. 舍入误差

当计算机执行算法时，由于受计算机的有效数字位数的限制，参加运算的数据总是只能具有有限位的有效数字，因而也就产生了舍人误差。

在数值计算过程中，由于计算机（别的计算工具也是如此）只能对数据的有限位数进行运算，因而在运算过程中不可避免地会产生误差。但是，如果能够掌握误差产生的规律，就可以将误差限制在最小的范围之内。而实际上，在运算过程中所产生的误差的大小，通常又与运算步骤有关。例如，在计算机上进行算术运算时，$a+b+c$ 可能不等于 $a+c+b$，同样（$a+b$）c 也可能不等于 $ac+bc$。

2.5.2　误差处理的几种方法

如上所述，计算同样的问题，不同的顺序、不同的计算形式所产生的误差大小是不一样的，所以在可能会出现较大误差的地方，我们可以通过各种手段尽量地减少误差程度。分析运算误差时，要考虑的原则如下：

（1）两个相近的数相减，会严重丢失有效字。

假设 $$y=x-A$$
其中 x 和 A 均为精确值。为了简单起见，假设 A 在运算时不产生误差，而 x 有误差，其近似值为 x^*。

$$E_r(y)=\frac{E(y)}{y^*}=\frac{(x-A)-(x^*-A)}{x^*-A}=\frac{x-x^*}{x^*-A}=\frac{E(x)}{x^*-A} \tag{2-4}$$

由上式可以看出，当 x 的绝对误差不变时，x^* 与 A 越接近，则 y 的相对误差的绝对值就越大。

两个相近的数相减，会产生极大的误差，但我们可以改变公式的形式，以减少误差。常用的公式变换方法如下：

1）当 x_1 很接近 x_2 时

$$\lg x_1-\lg x_2=\lg(x_1/x_2)$$

2）当 x 接近 0 时

$$\frac{1-\cos x}{\sin x}=\frac{\sin x}{1+\cos x}$$

3）当 x 充分大时

$$\arctan(x+1)-\arctan(x)=\arctan\frac{1}{1+x(x+1)}$$

$$\sqrt{x+1}-\sqrt{x}=\frac{1}{\sqrt{x+1}+\sqrt{x}}$$

两个相近的（比较复杂）函数值相减的时候，可用 Taylor 展开计算。

（2）除数绝对值较小时，商的绝对值会增大。

假设

$$z=\frac{x}{y}$$

其中 x 和 y 的近似值分别为 x^* 和 y^*，则

$$z^*=\frac{x^*}{y^*}$$

由此可得到 z 的绝对误差为：

$$E(z)=z-z^*=\frac{x}{y}-\frac{x^*}{y^*}=\frac{y^*x-y^*x^*-x^*y+x^*y^*}{yy^*}$$
$$=\frac{y^*(x-x^*)-x^*(y-y^*)}{yy^*}=\frac{y^*E(x)-x^*E(y)}{(y^*)^2} \tag{2-5}$$

由上面的式子可以看出，当两个数相除时，分母（即除数）的绝对值越小，商的绝对值就越大。

（3）在运算过程中必须注意合理安排运算顺序，以便提高运算精度或保护重要的参数。

在制定一个计算步骤时，还必须注意某些重要参数是否被“吃掉”的问题。例如：

$$110000.00-100000.00-10000.001=10000.000-10000.001=-0.001$$

假设在一个只有 8 位有效数字的计算机上计算，则

$$110000.00-10000.001-100000.00=100000.00\ (1)\ -100000.00=0$$

由此可以看出，在作运算时，应事先分析一下参与运算的各数值的数量级，然后合理安排它们的运算顺序，这样，一些重要的参数就不至于在运算过程中被其他参数“吃掉”。特别是在作连加运算时，合理安排它们的运算顺序，可以得到精度较高的结果。

（4）注意计算步骤的简化，减少算术运算的次数。

如前所述，运算过程中的每一步都有可能产生误差，而且这些误差还有可能传到下一步去，这种传递有时是增大的，有时是减小的。同时，运算过程中的每一步误差，也会积累到最终的结果中去，只不过这种误差的积累有时是增加的，有时则因互相抵消而减小。总而言之，在运算过程中都有可能引起导致误差增大的误差传播或误差积累等问题。因此，减少运算次数，就减少了产生误差的机会，也使误差积累有可能减小。

2.5.3　病态函数的判别

设有函数

$$y=f(x),a\leqslant x\leqslant b$$

如果在计算函数值 $f(x)$ 时，用 x 的近似值 x^* 来代替，那么函数 $f(x)$ 就被 $f(x^*)$ 所代替，其绝对误差为：

$$E[f(x)]=f(x)-f(x^*)$$

通常，我们总是希望当 x^* 很接近 x 时，$f(x^*)$ 也能很好地接近 $f(x)$，这时 $E[f(x)]$ 会很小。但是，实际情况可能不是这样。对于某些 x^*，$f(x^*)$ 也能很好地接近 $f(x)$，而对于另外一些 x^* 则不能，这样的函数我们称为病态函数。这主要取决于 $f(x)$ 的特性。例如分段函数、一些分数形式的函数中，就存在这样的情形。

2.5.4　算法的稳定性

在实际计算中，参与运算的各种数一般都带有一定的误差，这个误差或者是初值本身就有的（例如截断误差），或者是由于受计算机有效数字位数的限制所

造成的舍入误差。这些初始数据的误差（也称为摄动）以及在运算过程中产生的误差即使很小，但随着计算过程的进行，这些误差也会不断地传播下去，对以后的结果产生一定的影响。所谓稳定性问题，是指误差的传播（或积累）是否受控制的问题。如果计算结果对初始数据的误差以及计算过程中的舍入误差不敏感，则可认为相应的算法是稳定的，否则就称之为不稳定的。

例如，$x_0=a+\varepsilon$，其中 ε 为误差，且 $\varepsilon<o(a)$，如果迭代过程为：

$$x_0=a+\varepsilon$$

$$x_n=x_{n-1}^2,\ n=1,\ 2,\ \cdots$$

在这样的迭代过程中，误差 ε 对结果的影响是一个逐渐增大的过程。如果迭代过程为：

$$x_0=a+\varepsilon>0$$

$$x_n=\sqrt{x_{n-1}},\ n=1,\ 2,\ \cdots$$

在这样的迭代过程中，误差 ε 对结果的影响是一个逐渐缩小的过程。



第3章　MATLAB 基本介绍

对于计算机语言，只有正确合理地输入编译器才能正确执行函数，并输出我们想要的结果，MATLAB 也不例外。本章主要对 MATLAB 的背景以及基本使用方法作简单介绍，以便能正确地使用 MATLAB 进行运筹学与最优化问题的求解。

3.1　MATLAB 的发展历程和影响

MATLAB 由 MATrix 和 LABoratory 两词的前三个字母组合而成。那是 20 世纪 70 年代后期的事情：时任美国新墨西哥大学计算机科学系主任的 Cleve Moler 教授出于减轻学生编程负担的动机，为学生设计了一组调用 LINPACK 和 EISPACK 库程序的"通俗易用"的接口，此即用 FORTRAN 语言编写的萌芽状态的 MATLAB。经过几年的校际流传，在 Little 的推动下，由 Little、Moler、Steve Bangert 合作，于 1984 年成立了 MathWorks 公司，并把 MATLAB 正式推向市场。从那时起，MATLAB 的内核采用 C 语言编写，而且除原有的数值计算能力外，还新增了数据图视功能。MATLAB 以商品形式出现后的短短几年，就以其良好的开放性和运行的可靠性，使原先控制领域里的封闭式软件包纷纷淘汰，而改在 MATLAB 平台上重建。到了 20 世纪 90 年代，MATLAB 已经成为国际控制界公认的标准计算软件。90 年代初期，在国际上 30 几个数学类科技应用软件中，MATLAB 在数值计算方面独占鳌头，而 Mathematica 和 Maple 则分居符号计算软件的前两名。Mathcad 因其提供计算、图形、文字处理的统一环境而深受中学生欢迎。

MathWorks 公司于 1993 年推出了基于 Windows 平台的 MATLAB4.0。4.x 版在继承和发展其原有的数值计算和图形可视能力的同时，出现了以下几个重要变化：①推出了 SIMULINK——一个交互式操作的动态系统建模、仿真、分析集成环境。②推出了符号计算工具包——一个以 Maple 为"引擎"的 Symbolic Math Toolbox 1.0。此举结束了国际上数值计算、符号计算孰优孰劣的长期争论，促成了两种计算的互补发展新时代。③构作了 Notebook 。MathWorks 公司瞄准应用范围最广的 Word，运用 DDE 和 OLE，实现了 MATLAB 与 Word 的无缝连接，从而为专业科技工作者创造了融合科学计算、图形可视、文字处理于一体的高水准环境。从 1997 年春的 5.0 版起，后历经 5.1、5.2、5.3、6.0、6.1 等多个版本的不断改进，MATLAB"面向对象"的特点愈加突出，数据类型愈加丰富，操作界

面愈加友善。2002 年初夏推出的 6.5 版的最大特点是：采用了 JIT 加速器，从而使 MATLAB 朝着运算速度与 C 程序相比肩的方向前进了一大步。从 2006 年开始，MathWorks 公司宣布每年更新两次版本，已经有了 MATLAB2006a，MATLAB2006b，MATLAB2007a，MATLAB2007b，…，MATAB2009a 等。

本书中的所有程序都经 MATLAB2008a 测试计算。

3.2 MATLAB 界面介绍

MATLAB 界面如图 3-1 所示。

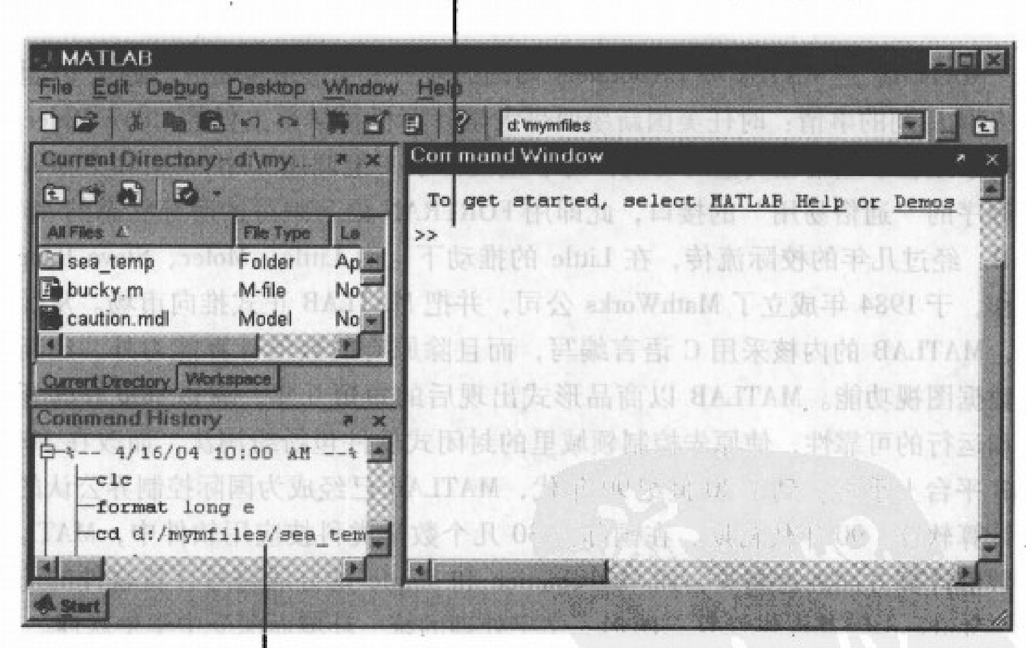


图 3-1

一般情况下 MATLAB 的初始界面主要由以下四部分组成：

(1) Command Window（命令行界面）：包含主要的数值计算、函数参数设定、函数调用及其结果输出。

(2) Command History（历史命令界面）：存储 Command Window 曾输入的历史命令。

(3) Current Directory（当前工作目录）：显示当前工作目录下的文件。

(4) Workspace（工作空间）：包含现实与计算相关的变量名称及其数值。

3.3 MATLAB 操作介绍

MATLAB 语法与函数众多，这里只介绍一些基本操作，其他具体函数可以参看 MATLAB 的帮助（help）。help 使用方法是在 Command window 输入 help + 函数名称。

例如输入：help rand

结果输出：

```
RAND    Uniformly distributed pseudo-random numbers.
    R = RAND(N) returns an N-by-N matrix containing pseudo-random values
    drawn from a uniform distribution on the unit interval.  RAND(M,N)
    or RAND([M,N]) returns an M-by-N matrix.  RAND(M,N,P,…) or
    RAND([M,N,P,…]) returns an M-by-N-by-P-by-… array. RAND with
    no arguments returns a scalar.  RAND(SIZE(A)) returns an array the
    same size as A.
        ⋮
    For a full description of the Mersenne Twister algorithm, see
    http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html.
```

本书不对具体的 MATLAB 基本语法进行介绍，以下只对 MATLAB 的常用知识点进行提示性介绍：

1. MATLAB 常用到的永久变量

(1) ans：计算结果的默认变量名。

(2) i j：基本虚数单位。

(3) eps：系统的浮点精度。

```
>> eps
     2.2204e-016
```

（系统计算的精度）

(4) inf：无限大，例 1/0。

(5) nan 或 NaN：非数值。

(6) pi：圆周率 π。

(7) realmax：系统所能表示的最大数值。

```
>> realmax
     1.7977e+308
```

(8) realmin：系统所能表示的最小数值。

```
>> realmin
        2.2251e-308
```

(9) nargin：函数的输入参数个数。

(10) nargout：函数的输出参数个数。

(11) MATLAB 的所有运算都定义在复数域上。对于方根运算只返回处于第一象限的解。

(12) MATLAB 分别用左斜“/”和右斜“\”来表示“左除”和“右除”运算。对于标量运算而言，这两者的作用没有区别：但对于矩阵运算来说，二者将产生不同的结果。具体可以参看高等代数中关于矩阵运算的概念。

2. 多项式的表示方法和运算

p(x) = x^3 - 3x + 5 可以表示为 p = [1 0 -3 5]，求 x = 5 时的值用 polyval (p,5)。

也可以求向量：a = [3 4 5]，polyval (p,a)

```
>> p = [1 0 -3 5]
```

表示三次项系数为 1，二次项系数为 0，一次项系数为 -3，常数项为 5，具体可以参看 MATLAB 的 Help 文档。

```
>> x = 5, polyval (p,x)
  115
>> a = [3 4 5], polyval (p,a)
  23     57     115 (计算 a 中每个元素对应多项式的值)
```

函数 roots 求多项式的根 roots(p)

```
>> p = [1 0 -3 5];
>> r = roots(p)
  r =
     -2.2790              (多项式的三个根)
      1.1395 + 0.9463i
      1.1395 - 0.9463i
```

有时 roots(p) 会产生虚根，这时用 real 函数可抽取实根

```
>> real(r);
ans =
   -2.2790
   1.1395
   1.1395
```

函数 conv(a,b) 为多项式乘法(执行两个数组的卷积)

```
>> a = [1 2 3 4];
>> b = [1 4 9 16];
>> c = conv(a,b)
  c =
       1    6    20    50    75    84    64
```

(即多项式为:x^6 +6x^5 +20x^4 +50x^3 +75x^2 +84x +64)

多项式的加减法,低阶的多项式必须用首零填补,使其与高阶多项式有同样的阶次。

多项式除法 [q , r] = deconv(c , b) 表示 b/c,q 为商多项式,r 为余数。

多项式的导数 polyder(f)

```
>> f = [ 2 4 5 6 2 1];
>> s = polyder(f)
s =
       10    16    15    12    2
```

3. 多项式的曲线拟合

```
x = [1 2 3 4 5];
y = [5.6 40 150 250 498.9];
```

p = polyfit(x,y,n) 数据的 n 次多项式拟合 poly:矩阵的特征多项式、根集对应的多项式, n 取 1 时,即为最小二乘法(线性回归方程)。

```
>> x = [1 2 3 4 5];
>> y = [5.6 40 150 250 498.9];
>> p = polyfit(x,y,1)
p =119.6600  -170.0800
```

(第一个数值为一次项系数,另一个为常数项)

分析拟合结果:

```
>> x2 =1:0.1:5;
>> y2 = polyval(p,x2);
```

计算多项式的值 (polyvalm 计算矩阵多项式)

```
>> plot(x,y,'*',x2,y2);
```

plot 画函数曲线,如图 3-2 所示。

三次函数拟合示例:

```
>> x = [1 2 3 4 5];
>> y = [5.6 40 150 250 498.9];
>> p = polyfit(x,y,3)
p =6.1083   -25.0464   84.2452   -63.2000
```

分析拟合结果

```
>> x2 =1:0.1:5;
```

```
>> y2 = polyval(p,x2);
>> plot(x,y,'*',x2,y2);
```

结果如图 3-3 所示。

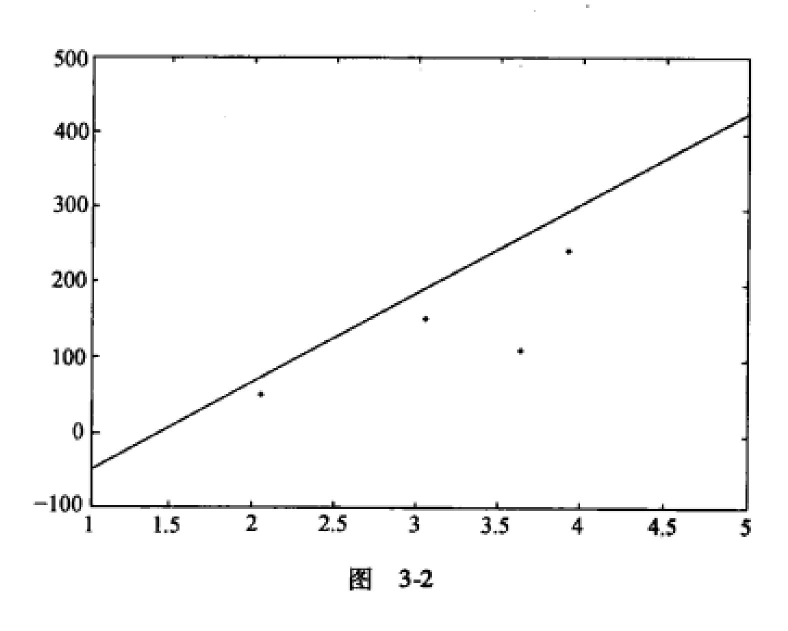


图 3-2

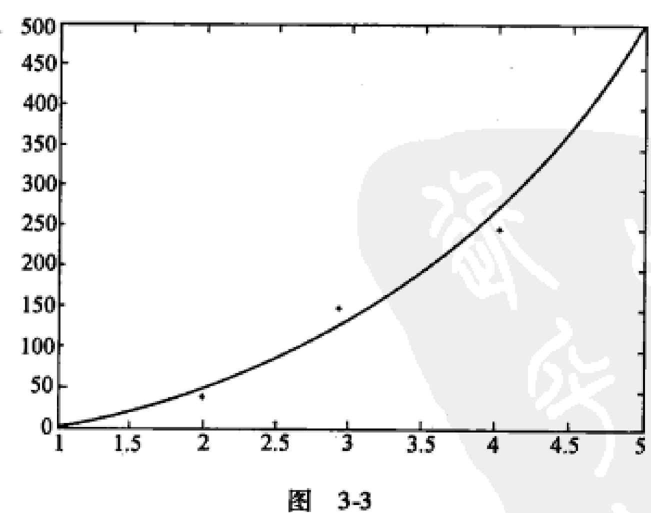


图 3-3

MATLAB 提供了曲线拟合工具 CFTOOL,含有更多的拟合模型,具体可以参看该函数说明。

4. 多项式插值

一维插值函数 YI = interp1(x,y,XI,'method')

其中,XI 为插值点的自变量坐标向量,可以为数组或单个数。

method为选择插值算法的方法,包括:linear(线性插值)、cubic(立方插值)、spline(三次样条插值)、nearst(最近邻插值)等。

例如,人口预测:

year = 1900:10:2000;

number = [78 91 105 …. 每10年的人口数];

x = 1900:1:2000;

y = interp1(year,number,x,'spline');　最后一个参数表示使用的插值方法

函数interp1(一维插值函数)提供的不同插值方法如下:

'nearest'　-最近邻插值

'linear'　-线性插值

'spline'　-分段3次样条插值(SPLINE)

'pchip'　-分段3次厄米多项式插值

例如,Nearest(最近邻插值法):

最近邻插值法(Nearest Neighbor)又称泰森多边形方法。泰森多边形(Thiessen,又叫Dirichlet或Voronoi多边形)分析法是荷兰气象学家A. H. Thiessen提出的一种分析方法,最初用于通过离散分布气象站的降雨量数据计算平均降雨量,现在GIS和地理分析中经常采用泰森多边形进行快速的赋值。实际上,最近邻插值法一个隐含的假设条件是,任一网格点p(x,y)的属性值都使用距它最近的位置点的属性值,用每一个网格节点的最邻点值作为待选的节点值。当数据已经是均匀间隔分布时,要先将数据转换为SURFER的网格文件,可以应用最近邻插值法;或者在一个文件中,数据紧密完整,只有少数点没有取值,可用最近邻插值法来填充无值的数据点。有时需要排除网格文件中的无值数据的区域,给搜索椭圆(Search-Ellipse)设置一个值,对无数据区域赋予该网格文件里的空白值。设置的搜索半径的大小要小于该网格文件数据值之间的距离,所有的无数据网格节点都被赋予空白值。在使用最近邻插值网格化法,将一个规则间隔的XYZ数据转换为一个网格文件时,可设置网格间隔和XYZ数据的数据点之间的间距相等。最近邻插值网格化法没有选项,它是均质且无变化的,对均匀间隔的数据进行插值很有用,同时,它对填充无值数据的区域很有效。

在应用中,常遇到一维傅里叶变换插值。一维傅里叶变换插值使用函数interpft实现,计算含有周期函数值的矢量的傅里叶变换,然后使用更多的点进行傅里叶变换的逆变换。函数的使用格式如下:

y = interpft(x,n)

其中,x是含有周期函数值的矢量,并为等距的点,n为返同等间距点的个数。

示例：

```
>> y = [0 .5 1 1.5 2 1.5 1 .5 0 -.5 -1 -1.5 -2 -1.5 -1 -.5 0];
   N = length(y);
    >> L = 5;
   M = N * L;
   x = 0:L:L * N - 1;
   xi = 0:M - 1;
   yi = interpft(y,M);
   plot(x,y,'o',xi,yi,' * ');
legend('Original data','Interpolated data');
```

如图 3-4 所示。

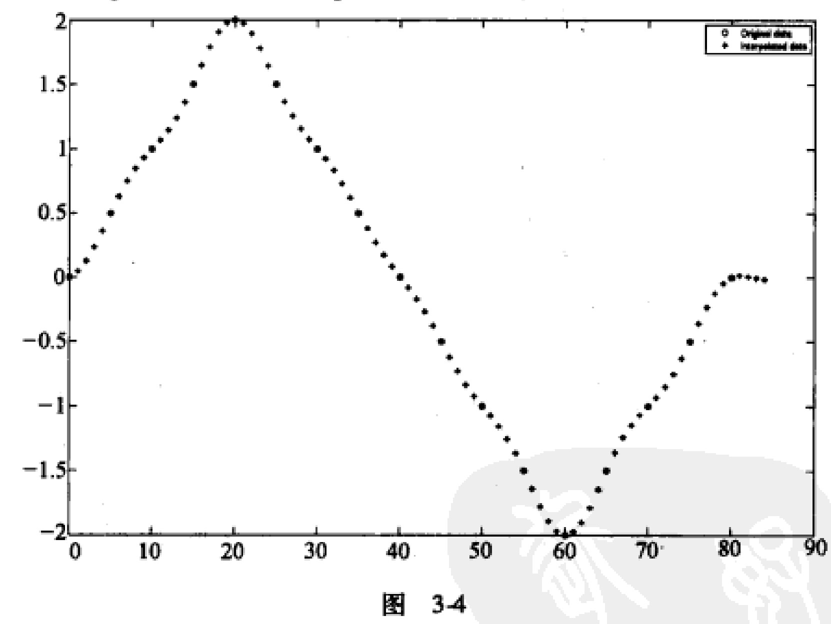


图 3-4

5. 求解一元函数的最小值

最优化为本书重点介绍部分，以下仅仅是简单示例：

测试函数 humps 为 MATLAB 一内置函数。

HUMPS A function used by QUADDEMO, ZERODEMO and FPLOTDEMO.

humps = [5.1765 9.1046 15.4706 26.4290 45.8868 76.1622 96.5000 76.9197 47.4483 28.8929 19.0000 13.9469 11.6923 11.3033 12.3824 14.7059 17.8462 20.7294 21.7027 19.8416 16.0000]

HUMPS 是一个 20 次函数，其曲线图像如图 3-5 所示。

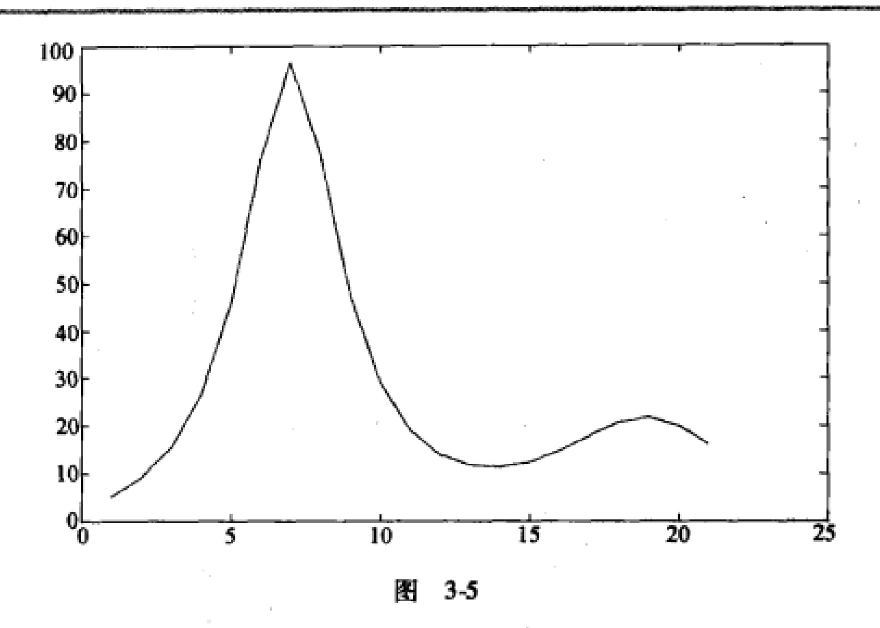


图　3-5

y = fminbnd('humps' ,0. 3,1)

求解'humps'函数在[0. 3,1]上的最小值对应的变量值。

y =0. 6370

函数 fminserch 用于求多元函数的最小值。它可以指定一个开始的矢量,并非指定一个区间。此函数返回一个矢量为此多元函数局部最小函数值对应的自变量。

6. 纹理成图功能

由 warp 函数的纹理成图功能实现平面图像在空间三维曲面上的显示。

将文件名为 flowers. tif 的图像分别投影到圆柱形和球形表面上。

```
i = imread( 'flowers. tif' ) ;
[x,y,z] = cylinder;
subplot(1,2,1),warp(x,y,z,i);
[x,y,z] = sphere(50);
subplot(1,2,2),warp(x,y,z,i);
warp(x,y,z,i);
```

注释:flowers. tif 需要自定义图片。

7. 求函数的零点

```
>> polyf1 = [ 0.3049    0.4084    -0.2681    -0.2607    -0.4502];
```

定义一个多项式。

```
>> [X,FVAL,EXITFLAG] = fzero(@(x)polyval(polyf1,x),[1,2]);
```

求函数 polyfl 在[1,2]区间上的零值点。

X = 1.1303 (0 值点对应的变量值)

FVAL = 1.1102e-016 (0 值点对应的函数值)

EXITFLAG = 1 (程序结束标记)

也可以给一个初始值，计算以该初始值为初始迭代点，得到函数值为零的变量值及实际函数值。

```
>> [X,FVAL,EXITFLAG] = fzero(@ (x),polyval(polyployfl,x),0.9);
```

X = 1.1303 (0 值点对应的变量值)

FVAL = 1.1102e-016 (0 值点对应的函数值)

EXITFLAG = 1 (程序结束标记)

对于多项式可直接由 roots 求其根。

```
>> roots(polyfl)
ans =
   -1.8124
    1.1303
   -0.3287 +0.7828i
   -0.3287 -0.7828i
```

8. 函数定积分

计算 $f(x)=x^3-2x-5$ 在[0,2]上的积分可以使用 quad。

```
>> F = @(x)1./(x.^3-2*x-5);
>> Q = quad(F,0,2)
Q = -0.4605
```

对于二重积分，首先计算内积分，然后借助内积分的中间结果再求出二重积分的值，类似于积分中的分步积分法。

例如，计算 $F=\int_{\pi}^{2}\int_{0}^{\pi}(y\sin x+x\cos y)\mathrm{d}x\mathrm{d}y$。

```
>> F = @(x,y)y*sin(x)+x*cos(y);
>> Q = dblquad(F,pi,2*pi,0,pi);
  Q = -9.8696
```

符号积分运算 int(f)

最精确的是符号积分法。例如，计算 $s=\int_{1}^{2\pi}\int_{0}^{1}xy\mathrm{d}x\mathrm{d}y$。

```
>> syms x y% 中间为空格,不能为逗号
>> s = int(int('x*y','x',0,1),'y',1,2)% 引号可省略
```

s = 3/4

9. 微分运算(diff)

微分是描述一个函数在一点处的斜率，是函数的微观性质，因此积分对函数的形状在小范围内的改变不敏感，而微分很敏感。一个函数的小的变化，容易产生相邻点的斜率的大的改变。由于微分这个固有的困难，所以要尽可能避免数值微分，特别是对实验获得的数据进行微分。在这种情况下，最好用最小二乘曲线拟合这种数据，然后对所得到的多项式进行微分；或用另一种方法对点数据进行三次样条拟合，然后寻找样条微分。但是，有时微分运算是不可避免的，在 MATLAB 中，可用函数 diff 计算一个矢量或者矩阵的微分（也可以理解为差分）。

```
>> a = [1 2 3 3 3 7 8 9];
>> b = diff(a) % 一次微分
  b =  1    1    0    0    4    1    1
>> bb = diff(a,2) % 二次微分
  bb = 0    -1    0    4    -3    0
```

实际上 diff(a) = [a(2) - a(1), a(3) - a(2), …, a(n) - a(n - 1)]

对于求矩阵的微分，即为求各列矢量的微分，从矢量的微分值可以判断矢量的单调性、是否等间距以及是否有重复的元素。

符号微分运算(diff)，有下列形式：设 f = cos(a * x)，

df = diff(f) % 由 findsym 的规则，隐含指定对 x 进行微分，

dfa = diff(f, 'a') % 指定对变量 a 进行微分，

dfa = diff(f, 'a', 3) % 指定对变量 a 进行三次微分。

diff 函数不仅作用在标量上，还可以在矩阵上，运算规则就是按矩阵的元素分别进行微分。

10. 计算多元函数的梯度

函数 fx = gradient(f)，f 是一个矢量返回 f 的一维数值梯度，fx 对应于 x 方向的微分。

例如：

```
[x,y] = meshgrid( -2:.2:2, -2:.2:2);
z = x. * exp( -x.^2 - y.^2);
[px,py] = gradient(z,.2,.2);
contour(z), hold on; % 画等值线。
quiver(px,py) % 如图 3-6 所示。
```

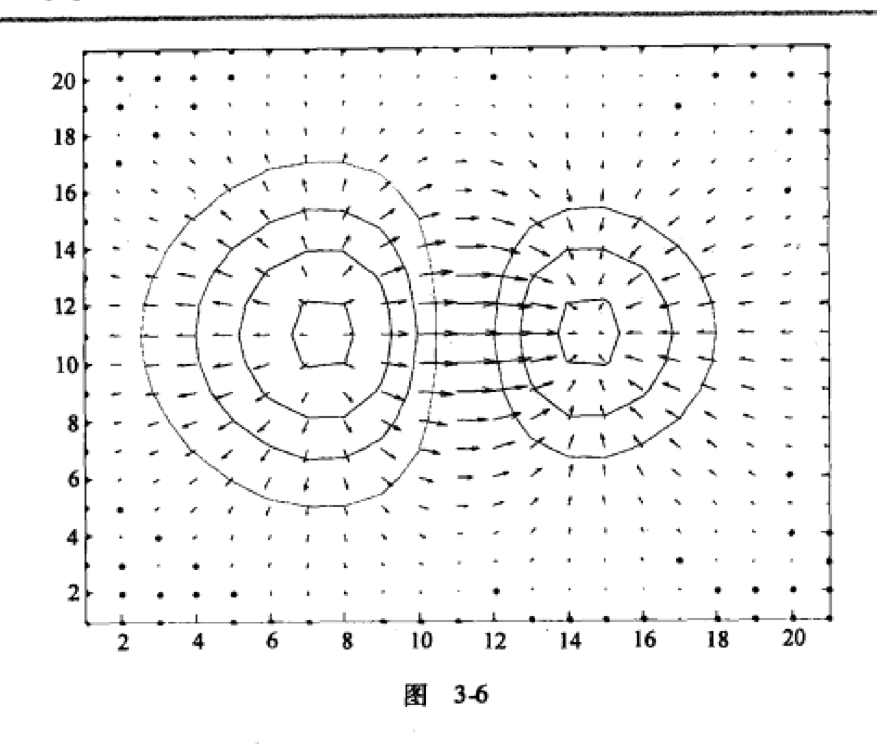


图　3-6

11. MATLAB 字符串运算

利用 sym 命令创建表达式。

f = sym('cos(x) + sin(x)')或 syms x ，f = cos(x) + sin(x)

diff(f) 求其导数

(也可直接用命令 f = diff('cos(x) + cos(y)')

当字符表达式中含有多于一个的变量时，只有一个变量是独立变量。如果不告诉 MATLAB 哪一个变量是独立变量，则可以通过 findsym 命令询问。

利用 findsym 命令查询独立变量：

f = sym('sin(a * x) + b')

findsym(f,1) 给出独立变量(一个变量，如果为 2 则给出 2 个变量)

findsym(f) 给出所有变量

符号表达式的化简和替换：

collect 函数：　collect(f,v)表示将 f 表示为关于符号变量 v 的多项式形式，即关于 v 合并同类项；v 缺省，则用 findsym 确定的缺省变量

syms x y

f = x^2 * y + y * x - x^2 - 2 * x + 1

collect(f) 得到(-1 + y) * x^2 + (y - 2) * x + 1

collect(f,y) 得到(x + x^2) * y + 1 - x^2 - 2 * x

expand 函数：　expand(f)将 f 展开，写成和的形式

syms x

expand((x-1)^3) 得到 x^3-3*x^2+3*x-1

horner 函数：　horner(f)将 f 写成镶嵌套形式

syms x

horner(x^3-6*x^2) 得到(-6+x)*x^2

factor 函数：　factor(f)将 f 转换成低阶有理多项式的乘积

syms x

f=x^3-6*x^2+11*x-6

factor(f) 得到 (x-1)*(x-2)*(x-3)

simplify(f)函数：　综合化简

simple(f) 函数的最简形式

syms x

f=2*sin(x^2)+cos(3*x)

simple(f) 如果不想看到中间过程，可用 z=simple(f) 有时使用两次 simple 命令可以得到最简式。

如果想知道哪个简化命令得到最后结果，可以加一个参数 how，例如：

[z,how]=simple(f)

符号表达式的替换：

subs(f,new,old)表示用 new 替换函数 f 中的变量 old，例如：

f='a*x^2+b*x+c'

subs(f,'t','x') 得到 a*(t)^2+b*(t)+c，subs 是一个符号函数，返回一个符号变量

subexpr 函数，有时 MATLAB 返回的符号表达式难以理解，用 subexpr 函数可以将表达式中重复出现的子式用一个符号(一般标示为 sigma)表示，从而简化表达形式，例如：

c=sym('c','real');

x=sym('x','real');

s=solve(x^3-x+c);

a=subexpr(s) 得到 sigma = -108*c+12*(-12+81*c^2)^(1/2)

a =

[1/6*sigma^(1/3)+2/sigma^(1/3)]

[-1/12*sigma^(1/3)-1/sigma^(1/3)+1/2*i*3^(1/2)*(1/6*sigma^(1/3)-2/sigma^(1/3))]

[-1/12 * sigma^(1/3) -1/sigma^(1/3) -1/2 * i * 3^(1/2) * (1/6 * sigma^(1/3) -2/sigma^(1/3))]

12. 线性方程组的求解

求解线性方程组，用反斜杠\。ax = b -> x = a\b (b 左除 a)，例如

```
>> a = hilb(3)
a =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
>> b = [1 2 3]'
>> x = a\b
x =
     27.0000
   -192.0000
    210.0000
```

13. 矩阵的特征值和特征向量

用 eig(v,d) 函数，[v,d] = eig(A)；其中 d 将返回特征值，v 返回相应的特征向量，缺省第二个参数将只返回特征值。例如：

```
syms a b c real
A = [a b c; b c a; c a b];
[v,d] = eig(A);
```

为了观察更清楚，使用以前学过的替换函数，这里不用默认的 sigma，而改用 m，显式的代替繁琐的表达子式。

vv = subexpr(v);

vs = subs(vv,'m','sigma') 运行结果为

```
vs =
[ 1, 1, 1]
[ -(c+(m)-a)/(c-b), -(c-(m)-a)/(c-b), 1]
[ -(a-(m)-b)/(c-b), -(a+(m)-b)/(c-b), 1]
```

再用 m 替换 d 中的表达子式：

dd = subexpr(d);

ds = subs(dd,'m','sigma')

运行结果为 ds =

[(m), 0, 0]

[0, -(m), 0]

[0, 0, c+a+b]

note 求特征值也可用以下命令：

f=poly(A) poly 函数 用来求 A 的特征多项式

d=solve(f) solve(f)函数 用来求多项式的解

svd( )函数 求矩阵的奇异值分解，将矩阵分解为两个正交矩阵和对角矩阵的乘积

a=sym(hilb(2))

[u,s,v]=svd(a)

14. 代数方程和方程组

代数方程的求解可用 solve(f)命令，如果 f 不含 =，MATLAB 将给表达式置零。方程的未知量在默认的情况下由 findsym 决定或显式指出。

syms a b c x

solve(a*x^2+b*x+c) 以 x 为默认变量

solve(a*x^2+b*x+c,a) 指定 a 为变量

求含有等号的方程的解(一定要加单引号)：

f=solve('cos(x)=sin(x)')

x=solve('exp(x)=tan(x)') 如果不能求得符号解，就计算可变精度解。

求解方程组与单方程类似。例如，解一个三元一次方程

v=solve('a*u^2+v^2','u-v-1','a^2-5*a+6')

结果为 v =

a: [4x1 sym] u: [4x1 sym] v: [4x1 sym]

15. 一些常用的符号运算

极限运算 limit：

limit(f) 求 x 到 0 的极限

limit(f,x,a)或 limit(f,a) 求 x 到 a 的极限

limit(f,a,'left') limit(f,a,'right') 求 x 到 a 的左极限和右极限

limit(f,inf) 求 x 趋于无穷的极限

符号求和 symsum(s)：

symsum(s) 以默认的 findsym 决定的变量求和

symsum(s,v) 以 s 中指定的变量 v 求和

symsum(s,a,b) symsum(s,v,a,b) 从 a 到 b 的有限项求和

syms k n

symsum(k) 从 0 到 k 求和

symsum(k,0,n-1) 从 0 到 n-1 求和

symsum(1/k^2,1,inf) 无限项求和

16. 泰勒级数 taylor(f)

taylor(f)表示求 f 的 5 阶 talor 展开,可以增加参数指定展开的阶数(默认是 5),也可以对于多元函数指定展开的变量,还可以指定在哪个点展开。

```
syms x t
taylor(exp(-x))
taylor(log(x),6,1) 在 1 点的 6 阶 taylor 展开
taylor(x^t,3,t) 对 t 的 3 阶 taylor 展开
```

17. 积分变换

fourier 变换和逆变换 fourier(f):

fourier 分析可以将信号转换为不同频率的正弦曲线;可对离散数据进行分析,也可对连续时间系统进行分析,特别在信号和图形处理领域。离散变换(DFT)作用于有限数据的采集,最有效的是快速 fourier 变换(FFT)。

F = fourier(f) 独立变量 x,返回关于参数 w 的函数

F = fourier(f,v) 返回函数 F 关于符号对象 v 的函数

F = fourier(f,u,v) 对关于 u 的函数 f 进行变换,而不是缺省的 w,返回函数 F 是关于 v 的函数

```
syms t v w x
fourier(1/t)
fourier(exp(-t)*sym('Heaviside(t)'),v)
fourier(diff(sym('F(x)')),x,w)
```

Fourier 逆变换:

f = ifourier(F) 缺省独立变量 w,返回关于 x 的函数对 w 进行积分

f = ifourier(F,v) 返回函数 f 是关于符号对象 v 的函数,而不是缺省的 x

f = ifourier(F,u,v)是对关于 u 的函数 f 进行变换,而不是缺省的 x,返回函数 f 是关于 v 的函数

Laplace 变换和逆变换 laplace(f):

应用于连续系统(微分方程)中,可以用来求解微分方程的初值问题

laplace(F) 缺省独立变量 t,缺省返回关于 s 的函数 L

laplace(F,t) 返回关于 t 的函数 L,而不是缺省的 s

laplace(F,w,z) 对函数 F 的自变量 w 积分,返回关于 z 的函数 L

逆变换:

F = ilaplace(L) 缺省独立变量 s,返回关于 t 的函数 F

F = ilaplace(L,y) 返回关于 y 的函数 F,而不是缺省的 t

F = ilaplace(L,y,x) 对函数 L 的自变量 y 积分,返回关于 x 的函数 F

Z - 变换和逆变换 ztrans(f) 标量符号 f 的 Z - 变换：

F = ztrans(f) 缺省独立变量 n,返回关于 z 的函数

F = ztrans(f,w) 返回关于符号变量 w 的函数 F,而不是缺省的 z

F = ztrans(f,k,w) 关于 k 的符号变量作 Z - 变换,返回关于符号变量 w 的函数

逆变换 iztrans(F)：

f = iztrans(F) 或(F,k)或 (F,w,k)

18. 符号绘图函数

符号函数简易绘图函数 ezplot(f)：

f 可以包含单个符号变量 x 的字符串或表达式,默认画图区间(-2pi,2pi),如果 f 包含 x 和 y,画出的图像是 f(x,y) =0 的图像,缺省区间是 -2pi < x < 2pi, -2pi < y < 2pi。那么,函数的意义为,Ezplot(f,xmin,xmax)或 ezplot(f,[xmin,xmax])绘制在 xmin < x < xmax 区间上的图像。

```
syms x t
ezplot('t * cos(t)','t * sin(t)',[0,4 * pi])
```

绘制符号图像函数 fplot(fun,lims,tol,'linespec',n),其中 lims = [xmin,xmax] 或[xmin,xmax,ymin,ymax];tol 为指定相对误差,默认 0.001;'linespec'为指定绘图的线型;n 为指定最少以 n+1 个点绘图。例如：

[x,y] = fplot(fun,lims,…) 只返回用来绘图的点,并不绘图,可以自己调用 plot(x,y)来绘制图形。

```
syms x
subplot(2,2,1),fplot('humps',[0,1])
f = 'abs(exp(x * (0:9)) * ones(10,1))'
subplot(2,2,2),fplot(f,[0,2 * pi])
subplot(2,2,3),fplot('sin(1./x)',[0.01,0.1],1e-3)
```

二维图形的绘制：

plot 在(x,y)坐标下绘制二维图像,支持多个 x - y 二元结构

plot3 在(x,y,z)坐标下绘制三维图形

loglog 在(x,y)对数坐标下绘制二维图形

semilogx 在 x 为对数坐标、y 为线性坐标的二维坐标中绘图

semilogy 在 x 为线性坐标、y 为对数坐标的二维坐标中绘图

plotyy 在有两个 y 轴的坐标下绘图

plot 的用法：

plot(x,y,'—rs','linewidth',2,'markeredgecolor','k',…
'markerfacecolor','g','markersize',10)

plotyy 的用法：

plotyy(x1,y1,x2,y2) 以 x1 为标准，左轴为 y 轴绘制 y1 向量；以 x2 为基准，右轴为 y 轴，绘制 y2 向量

plotyy(x1,y1,x2,y2,fun)　用字符串 fun 指定的绘图函数(plot ,semilogx, semilogy, loglog, stem)

plotyy((x1,y1,x2,y2,fun1,fun2)

t=0:pi/20:2*pi;

y=exp(sin(t));

plotyy(t,y,t,y,'plot','stem')% stem 为二维杆图

[ax,h1,h2]=plotyy(…) 返回左右两 y 轴的句柄(分别为 ax(1),ax(2))，以及在两坐标轴中生成的图形对象的句柄，分别为 h1,h2

t=0:900;

A=1000;

a=0.005;

b=0.005;

z2=cos(b*t);

z1=A*exp(-a*t);

[haxes,hline1,hline2]=plotyy(t,z1,t,z2,'semilogy','plot');

axes(haxes(1))

ylabel('semilog plot') 对数坐标

axes(haxes(2))

ylabel('linear plot')

set(hline2,'linestyle','—')

其他二维图形绘图指令：

bar(x,y) 二维条形图

hist(y,n) 直方图

histfit(y,n) 带拟合线的直方图，n 为直方的个数

stem(x,y) 火柴杆图

comet(x,y) 彗星状轨迹图

compass(x,y) 罗盘图

errorbar(x,y,l,u) 误差限图

feather(x,y) 羽毛状图

fill(x,y,'r') 二维填充函数,以红色填充

pie(x) 饼图

polar(t,r) 极坐标图,r 为幅值向量,t 为角度向量

t=0:0.1:8*pi;

r=cos(3*t/2)+1/2;

polar(t,r),xlabel('polar 指令')

quiver(x,y) 磁力线图

stairs(x,y) 阶梯图

loglog(x,y) 对数图

semilogx semilogy 半对数图

MATLAB 三维作图

plot3(x,y,z) 三维线条图

t=0:pi/50:15*pi;

plot3(sin(t),cos(t),t,'r*') 与 plot 相似

v=axis 返回各个轴的范围

text(0,0,0,'origin') 在某个坐标点加入文字

plot3 增加维数可以一次画多个图,使所有二维图形沿一个轴排列。

19. 三维网线图的绘制

mesh(x,y,z) 网格图

mesh(x,y,z,c) 四维作图,(x,y,z)代表空间三维,c 代表颜色维

mesh(…,'property name',property value,…) 设置曲面各属性的值

[x,y,z]=sphere(12);

mesh(x,y,z),hidden off 曲面设置为透明

meshc(x,y,z) 画网格图和基本的等值线图

meshz(x,y,z) 画包含零平面的网格图

waterfall(x,y,z) 与 mesh 一样,只是在效果上它的网格线只在 x 轴一个方向出现,呈瀑布状水线

两个变量的标量指令 meshgrid(x)或 meshgrid(x,y)将两个一维向量生成两个二维向量,以便进行 z=f(x,y)运算,算出 z 的所有值,z 为 x,y 的标量指令

[X,Y]=meshgrid(x),meshgrid(x,x)的简略式

[X,Y]=meshgrid(x,y)

[X,Y,Z]=meshgrid(x,y,z),用于三维图形的绘制

[x,y]=meshgrid([-2:0.1:2]);

z = x. * exp(- x. ^2 - y. ^2);

plot3(x,y,z)

surf(x,y,z,c) 着色表面图

surf(x,y,z) 隐含着 c = z

surf(z) 隐含着 x,y 的值为 surf 指令根据 z 的尺寸自动生成

surfc 画出具有基本等值线的曲面图

surfl 画出一个具有亮度的曲面图

shading flat 网线图的某整条线段或曲面图的某个贴片都着一种颜色

shading interp 某一线段或贴片上各点的颜色由线或片的顶端颜色经线性插值而得。曲面图不能设成网格图那样透明，但需要时，可以在孔洞处将数据设成 nun。

20. 等高线的绘制

在二维空间绘制等高线 contour

contour(x,y,z,n) 绘制 n 条等值线(n 可省略)

contour(x,y,z,v) 在向量 v 所指定的高度上绘制等高线(可省)

c = contour(x,y,z) 计算等值线的高度值

c = contourc(x,y,z,n) 计算 n 条等高线的 x - y 坐标数据

c = contourc(x,y,z,v) 计算向量 v 所指定的等高线的 x - y 坐标数据

clabel(c) 给 c 阵所表示的等高线加注高度标识

clabel(c,v) 给向量 v 所指定的等高线加注高度标识

clabel(c,'manual') 借助鼠标给点中的等高线加注高度标识

三维空间绘制等高线 contour3(x,y,z)

[x,y,z] = peaks(30);

contour3(x,y,z,16,'g')

21. 二元函数的伪彩图 pcolor(x,y,z)

它是指令 surf 的二维等效指令，代表伪彩色，可与 contour 单色等值线结合画彩色等值线图。

[x,y,z] = peaks(30);

pcolor(x,y,z); 伪彩色

shading interp 颜色插值，使颜色平均渐变

hold on,contour(x,y,z,20,'k')… 画等值线

colorbar('horiz') 水平颜色标尺

c = contour(x,y,z,8);

clabel(c) 标注等高线

22. 矢量场图(速度图)quiver

用于描述函数 z = f(x,y)在点(x,y)的梯度大小和方向

[X,Y] = meshgrid(x,y) X,Y 为 Z 阵元素的坐标矩阵

[U,V] = gradient(Z,dx,dy) U,V 分别为 Z 对 x、对 y 的导数,dx,dy 是 x,y 方向上的计算步长

quiver(X,Y,U,V,s,'linespec','filled') U,V 为必选项,决定矢量场图中各矢量的大小和方向,s 为指定所画箭头的大小;缺省时取 1,linespec 为字符串,指定合法的线形和彩色,filled 用于填充定义的绘图标识符。

```
[x,y] = meshgrid( -2:.2:2, -1:.15:1);
z = x.*exp( -y.^2);
[px,py] = gradient(z,.2,.15);
contour(x,y,z);
hold on,quiver(x,y,px,py),axis image
```

23. 多边形的填色 fill(x,y,c)

c 定义颜色字符串,可以是'r','b'等,也可以用 RGB 三色表示[r,g,b]值为 0 - 1 图形的四维表现。例如:

```
>> x =0:0.1:10;
>> y = sin(x);
>> fill([x,10],[y,0],'r')
```

结果如图 3-7 所示。

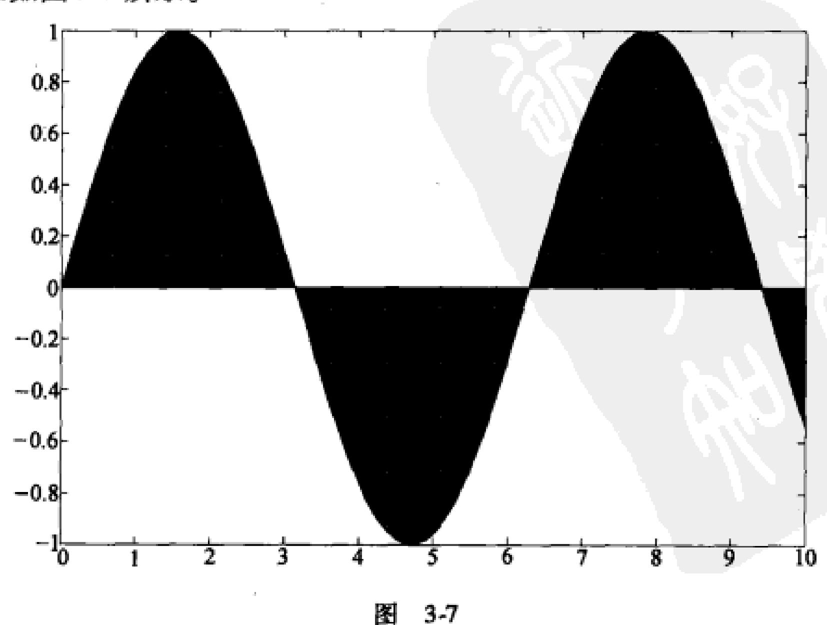


图 3-7

3.4 M 文件函数

使用 MATLAB 函数时，例如 inv，abs，angle 和 sqrt，MATLAB 获取传递给它的变量，利用所给的输入，计算所要求的结果；然后，把这些结果返回。由函数执行的命令以及由这些命令所创建的中间变量，都是隐含的。所有可见的东西是输入和输出，也就是说，函数是一个黑箱。

这些属性使得函数成为强有力的工具，用以计算命令。这些命令包括在求解一些大的问题时，经常出现的有用的数学函数或命令序列。由于这个强大的功能，MATLAB 提供了一个创建用户函数的结构，并以 M 文件的文本形式存储在计算机上。MATLAB 函数 fliplr 是一个 M 文件函数良好的例子。图 3-8 是 M 文件窗口。

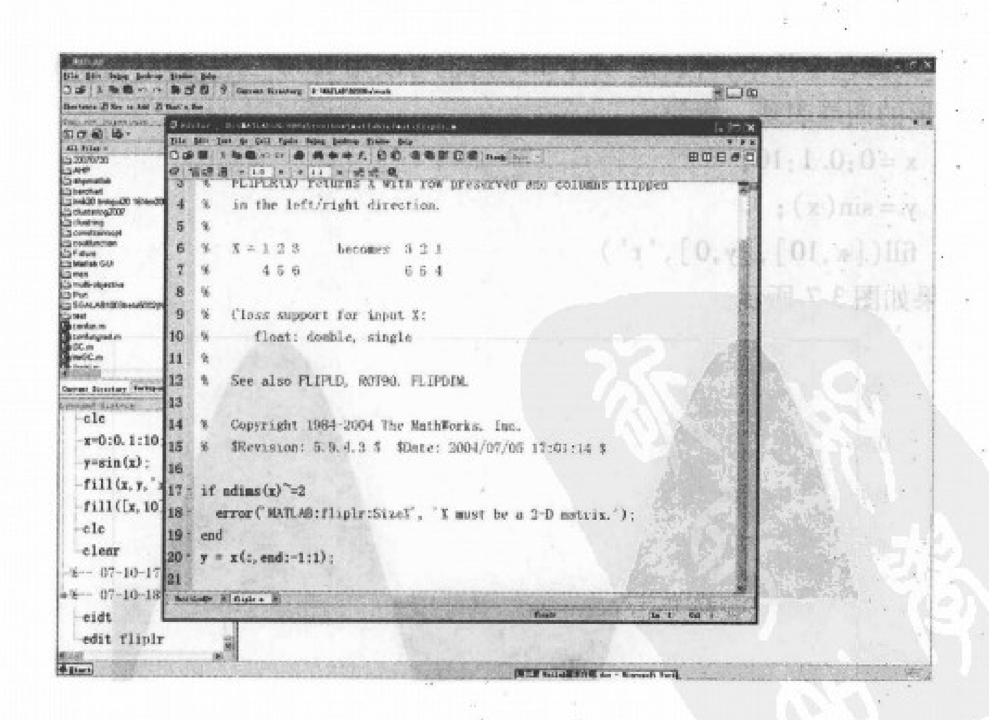


图 3-8

```
function y = fliplr(x)
%   FLIPLR   Flip matrix in the left/right direction.
```

```
%      FLIPLR(X) returns X with row preserved and columns flipped
%      in the left/right direction.
%
%      X = 1  2  3     becomes  3  2  1
%          4  5  6              6  5  4
%
%      See also FLIPUD, ROT90.
%      Copyright (c) 1984 -94 by The MathWorks, Inc.
[m, n] = size(x);
y = x(: , n : -1 : 1);
```

一个函数 M 文件与脚本文件类似之处在于它们都是一个有 .m 扩展名的文本文件。如同脚本 M 文件一样，函数 M 文件不进人命令窗口，而是由文本编辑器所创建的外部文本文件。一个函数的 M 文件与脚本文件在通信方面是不同的。函数与 MATLAB 工作空间之间的通信，只通过传递给它的变量和通过它所创建的输出变量。在函数内，中间变量不出现在 MATLAB 工作空间，或与 MATLAB 工作空间不交互。正如上面的例子所看到的，一个函数的 M 文件的第一行把 M 文件定义为一个函数，并指定它的名字。它与文件名相同，但没有 .m 扩展名。它也定义了它的输入和输出变量。接下来的注释行是所展示的文本，它与帮助命令：>> help fliplr 相对应。第一行帮助行称为 H1 行，是由 lookfor 命令所搜索的行。最后，M 文件的其余部分包含了 MATLAB 创建输出变量的命令。

M 文件函数必须遵循以下特定的规则。除此之外，它还有许多重要属性，包括：

（1）函数名和文件名必须相同。例如，函数 fliplr 存储在名为 fliplr. m 文件中。

（2）MATLAB 第一次执行一个 M 文件函数时，它打开相应的文本文件并将命令编辑成存储器的内部表示，以加速执行以后所有的调用。如果函数包含了对其他 M 文件函数的引用，它们也同样被编译到存储器里。普通的脚本 M 文件不被编译，即使它们是从函数 M 文件内调用；打开脚本 M 文件，调用一次就逐行进行注释。

（3）在函数 M 文件中，到第一个非注释行为止的注释行是帮助文本。当需要帮助时，返回该文本。例如，>> help fliplr 返回上述前八行注释。

（4）第一行帮助行，名为 H1 行，是由 lookfor 命令搜索的行。

（5）函数可以有零个或更多个输入参量。函数可以有零个或更多个输出参量。

（6）函数可以按少于函数 M 文件中所规定的输入和输出变量进行调用，但不能用多于函数 M 文件中所规定的输入和输出变量数目。如果输入和输出变量数目多于函数 M 文件中 function 语句一开始所规定的数目，则调用时自动返回一个错误。

（7）当函数有一个以上输出变量时，输出变量包含在括号内。例如，[V, D] = eig (A)。不要把这个句法与等号右边的 [V, D] 相混淆，右边的 [V, D] 是由数组 V 和 D 所组成。

（8）当调用一个函数时，所用的输入和输出参量的数目在函数内是规定好的。函数工作空间变量 nargin 包含输入参量个数；函数工作空间变量 nargout 包含输出参量个数。事实上，这些变量常用来设置缺省输入变量，并决定用户所希望的输出变量。例如，考虑 MATLAB 函数 linspace ：

```
function y = linspace(d1, d2, n)
%   LINSPACE Linearly spaced vector.
%       LINSPACE(x1, x2) generates a row vector of 100 linearly
%       equally spaced points between x1 and x2.
%       LINSPACE(x1, x2, N) generates N points between x1 and x2.
%
%       See also LOGSPACE, :.
%       Copyright (c) 1984 -94 by The MathWorks, Inc.
if nargin == 2
    n = 100;
end
y = [d1 + (0:n-2) * (d2 - d1)/(n-1) d2] ;
```

这里，如果用户只用两个输入参量调用 linspace ，例如 linspace(0, 10)，linspace 产生 100 个数据点。相反，如果输入参量的个数是 3，例如 linspace(0, 10, 50)，第三个参量决定数据点的个数。

可用一个或两个输出参量调用的函数的一个例子是 MATLAB 函数 SIZE。尽管这个函数不是一个 M 文件函数（它是一个内置函数），SIZE 函数的帮助文本说明了它的输出参量的选择。

```
SIZE   Matrix dimensions.
       D = SIZE(X), for M-by-N matrix X, returns the two-element
       row vector D = [M, N] containing the number of rows and columns
       in the matrix.
       [M, N] = SIZE(X) returns the number of rows and columns
```

in separate output variables.

如果函数仅用一个输出参量调用，就返回一个二元素的行，它包含行数和列数；相反，如果出现两个输出参量，size 分别返回行和列。在 M 文件函数里，变量 nargout 可用来检验输出参量的个数，并按要求修正输出变量的创建。

（9）当一个函数说明一个或多个输出变量，但没有要求输出时，就简单地不给输出变量赋任何值。MATLAB 函数 toc 阐明了这个属性。

```
function t = toc
%  TOC  Read the stopwatch timer.
%     TOC, by itself, prints the elapsed time since TIC was used.
%     t = TOC; saves the elapsed time in t, instead of printing it out.
%
%     See also TIC, ETIME, CLOCK, CPUTIME.
%     Copyright (c) 1984-94 by The MathWorks, Inc.
%  TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if nargout < 1
  elapsed_time = etime(clock, TICTOC)
else
  t = etime(clock, TICTOC);
end
```

如果用户不以输出参量调用 toc，例如，>>toc，就不指定输出变量 t 的值，函数在命令窗口显示函数工作空间变量 elapsed_time，但在 MATLAB 工作空间里不创建变量。相反，如果 toc 是以 >>out=toc 调用，则按变量 out 将消逝的时间返回到命令窗口。

（10）函数有它们自己的专用工作空间，与 MATLAB 的工作空间分开。函数内变量与 MATLAB 工作空间之间唯一的联系是函数的输入和输出变量。如果函数任一输入变量值发生变化，其变化仅在函数内出现，不影响 MATLAB 工作空间的变量。函数内所创建的变量只驻留在函数的工作空间，而且只在函数执行期间临时存在，以后就消失。因此，从一个调用到下一个调用，在函数工作空间变量存储信息是不可能的。（然而，如下所述，使用全局变量就可以提供这个特征）

（11）如果一个预定的变量，例如 pi，在 MATLAB 工作空间重新定义，它不会延伸到函数的工作空间。逆向有同样的属性，即函数内的重新定义变量不会延伸到 MATLAB 的工作空间中。

（12）当调用一个函数时，输入变量不会拷贝到函数的工作空间，但使它们的值在函数内可读。然而，改变输入变量内的任何值，那么数组就拷贝到函数工作空间。进而，按缺省，如果输出变量与输入变量相同，例如，函数 x = fun(x, y, z）中的 x，那么就将它拷贝到函数的工作空间。因此，为了节约存储和增加速度，最好是从大数组中抽取元素，然后对它们作修正，而不是使整个数组拷贝到函数的工作空间。

（13）如果变量说明是全局的，函数可以与其他函数、MATLAB 工作空间和递归调用本身共享变量。为了在函数内或 MATLAB 工作空间中访问全局变量，在每一个所希望的工作空间，变量必须说明是全局的。全局变量使用的例子可以在 MATLAB 函数 tic 和 toc 中看到，它们合在一起工作如一个跑表。

```
function tic
%  TIC  Start a stopwatch timer.
%      The sequence of commands
%          TIC
%          any stuff
%          TOC
%      prints the time required for the stuff.
%
%      See also TOC, CLOCK, ETIME, CPUTIME.
%      Copyright (c) 1984 - 94 by The MathWorks, Inc.
%  TIC simply stores CLOCK in a global variable.
global TICTOC
TICTOC = clock;
function t = toc
%  TOC  Read the stopwatch timer.
%      TOC, by itself, prints the elapsed time since TIC was used.
%      t = TOC; saves the elapsed time in t, instead of printing it out.
%
%      See also TIC, ETIME, CLOCK, CPUTIME.
%      Copyright (c) 1984 - 94 by The MathWorks, Inc.
%  TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if nargout < 1
   elapsed_time = etime(clock,TICTOC)
```

```
else
    t = etime(clock,TICTOC);
end
```

在函数 tic 中，变量 TICTOC 说明为全局的，因此它的值由调用函数 clock 来设定。以后在函数 toc 中，变量 TICTOC 也说明为全局的，让 toc 访问存储在 TICTOC 中的值。利用这个值，toc 计算自执行函数 tic 以来消逝的时间。值得注意的是，变量 TICTOC 存在于 tic 和 toc 的工作空间，而不是在 MATLAB 工作空间。

（14）实际编程中，无论什么时候应尽量避免使用全局变量。要是用了全局变量，建议全局变量名要长，它包含所有的大写字母，并有选择地以首次出现的 M 文件的名字开头。如果遵循建议，则在全局变量之间不必要的互作用将减至最小。例如，如果另一函数或 MATLAB 工作空间说明 TICTOC 为全局的，那么它的值在该函数或 MATLAB 工作空间内可被改变，而函数 toc 会得到不同的、可能是无意义的结果。

（15）MATLAB 以搜寻脚本文件的同样方式搜寻函数 M 文件。例如，输入 >> cow，MATLAB 首先认为 cow 是一个变量；如果它不是，那么 MATLAB 认为它是一个内置函数；如果还不是，MATLAB 检查当前 cow.m 的目录或文件夹。如果它不存在，MATLAB 就检查 cow.m 在 MATLAB 搜寻路径上的所有目录或文件夹。如需要更多的信息，请参阅本书的 2.10 节或 MATLAB 用户指南中"MATLAB 搜寻路径"。

（16）从函数 M 文件内可以调用脚本文件。在这种情况下，脚本文件查看函数工作空间，不查看 MATLAB 工作空间。从函数 M 文件内调用的脚本文件不必用调用函数编译到内存。函数每调用一次，它们就被打开和解释。因此，从函数 M 文件内调用脚本文件减慢了函数的执行。

（17）函数可以递归调用，即 M 文件函数能调用它们本身。例如，考虑下列函数 iforgot：

```
function iforgot(n)
%   IFORGOT Recursive Function Call Example
%   Copyright (c) 1996 by Prentice-Hall,Inc
if nargin == 0,n = 20;end
if n > 1
    disp(' I will remember to do my homework. ')
    iforgot(n - 1)
else
```

```
    disp(' Maybe NOT! ')
end
```

调用这个函数产生

```
>> iforgot(10)
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
Maybe NOT!
```

递归调用函数功能在许多应用场合是有用的。在编制要递归调用的函数时，必须确保会终止，否则 MATLAB 会陷入死循环。最后，在一个递归函数内，如果变量说明是全局的，则该全局变量对以后所有函数调用是可用的。在这个意义下，全局变量变成静态的，并在函数调用之间不会消失。

(18) 当函数 M 文件到达 M 文件终点，或者碰到返回命令 return，就结束执行和返回。return 命令提供了一种结束一个函数的简单方法，而不必到达文件的终点。

(19) MATLAB 函数 error 在命令窗口显示一个字符串，放弃函数执行，把控制权返回给键盘。这个函数对提示函数使用不当很有用，如在以下文件片段中：

```
if length (val) >1
    error ('  VAL must be a scalar.  ')
end
```

这里，如果变量 val 不是一个标量，error 显示消息字符串，把控制权返回给命令窗口和键盘。

(20) 当一个函数的输入参量的个数超出了规定的范围，MATLAB 函数 nargchk 提供了统一的响应。函数 nargchk 给定为：

```
function  msg = nargchk(low, high, number)
%  NARGCHK  Check number of input arguments.
%     Return error message if not between low and high.
%     If it is, return empty matrix.
```

```
%      Copyright (c) 1984 -94 by The MathWorks, Inc.
msg = [  ];
if (number < low)
      msg = ' Not enough input arguments. ';
elseif (number  >  high)
      msg = ' Too many input arguments. ';
end
```

下列的文件片段表明了在一个 M 文件函数内的典型用法：

```
error(nargchk(nargin, 2, 5))
```

如上所示，如果 nargin 的值小于 2，函数 error 像前面描述的那样进行处理，nargchk 返回字符串‘没有足够的输人参量。’如果 nargin 的值大于 5，函数 error 执行处理，nargchk 返回字符串‘太多输入参量’。如果 nargin 是在 2 和 5 之间，函数 error 简单地将控制传递给下一个语句，nargchk 返回一个空字符串。也就是说，当它的输入参量为空，error 函数什么也不做。

（21）当 MATLAB 运行时，它缓存了（caches）存储在 Toolbox 子目录和 Toolbox 目录内的所有子目录中的所有 M 文件的名字和位置。这使 MATLAB 能很快地找到和执行函数 M 文件，也使得命令 lookfor 工作更快。被缓存的 M 文件函数当作是只读的。如果执行这些函数，以后又发生变化，MATLAB 将只执行以前编译到内存中的函数，不管已改变的 M 文件。而且，在 MATLAB 执行后，如果 M 文件被加到 Toolbox 目录中，那么它们将不出现在缓存里，因此不可利用。所以，在 M 文件函数的使用中，最好把它们存储在 Toolbox 目录外，或许最好存储在 MATLAB 目录下，直至它们被认为是完备的（complete）。当它们被认为是完备的时，就将它们移到一个只读的 Toolbox 目录或文件夹的子目录内。最后，要确保 MATLAB 搜索路径改变，以确认它们的存在。

（22）在 Toolbox 目录外，MATLAB 跟踪 M 文件的修改日期。所以，当遇到一个以前编译到内存的 M 文件函数时，MATLAB 把已编译的 M 文件的修改日期与在磁盘上的 M 文件比较。如果日期是相同的，MATLAB 执行已编译的 M 文件。相反，如果在磁盘上的 M 文件是新的，MATLAB 清除以前已编译的 M 文件，且编译这个新的和修改过的 M 文件。

（23）M 文件的缓存过程因 MATLAB 版本的不同而稍有不同。例如，MATLAB4.2c 在 Macintosh 机上同样可以缓存当前的目录，因为这是第一个所搜索的磁盘位置。这个 MATLAB 版本也允许有选择地将整个 MATLAB 搜索路径缓存，并把高速缓存信息存储在一个文件中。这样，使 MATLAB 引导得更快，即寻找和编译所有函数 M 文件更快。退出缓存，不检测已修改的或已增加的 M 文件。

当新的 M 文件加到一个缓存区时，只有当高速缓存由命令 >> path 刷新时，MATLAB 才能找到它们；当修改缓存的 M 文件时，只有当以前编译过的版本由 clear 命令从内存中清除，MATLAB 才识别这个变化。例如， >> clear myfun，从内存中清除 M 文件函数 myfun，或 >> clear functions,从内存中清除所有已编译的函数。

（24）在变量 mfilename 函数内，有要执行的 M 文件的名字。例如，正在执行 M 文件 function. m 时，函数的工作空间包含变量 mfilename，它包含函数字符串。这个变量也存在于脚本文件里，在这种情况下，它包含了要执行的脚本文件的名字。

（25）M 文件函数可像 MATLAB 命令一样工作，典型的 MATLAB 命令包括 clear，disp，echo，diary，save，hold，load，more 和 format。通常，调用一个函数把参量放在括号内，例如 size(A)。然而,如果函数有字符串参量,那么,函数可按通常函数进行调用,如,disp('To be or not to be'),或像一个 MATLAB 命令来使用,如 clear functions。换句话说,当要求 MATLAB 解释一个表达式 >> command argument 时,MATLAB 认为它如同 >> command('argument')一样。事实上,MATLAB 命令本身能像函数那样调用,例如 >> format long 和 >> format('long')二者都把数据变成长格式。类似地，>> format short e 等价于 >> format('short','e')。正如最后的例子所示,空格(逗号,分号)把各个命令参量分开。因此，>> disp How about this? 产生一个错误,因为命令 disp 只允许一个输入参量,不是三个。如果参量包含在引号里,那么 MATLAB 就忽略空格,例如，>> disp' How about this?' 与 >> disp('How about this?')等价，并产生所希望的结果。

总之，函数 M 文件提供了一个简单的扩展 MATLAB 功能的方法。事实上，MATLAB 本身的许多标准函数就是 M 文件函数。

3.5 Excel-Link

MATLAB 提供使其能与 Excel 互动操作的 Excel-link 宏。Excel-link 使得数据在 MATLAB 与 Excel 之间随意交换（如图 3-9 所示），以及在 Excel 下调用 MATLAB 的函数。Excel-link 将 MATLAB 强大的数值计算功能、数据可视画功能与 Excel 的数据 Sheet 功能结合在一起。下面就简单介绍 Excel-link 的基本操作。

1. 加载 Excel-link 宏

加载方法：在 Excel 工具上→加载宏→浏览（MATLAB 的安装路径）→toolbox 文件夹→exlink 文件夹→excllink. xla 文件（打开），如图 3-10 所示。

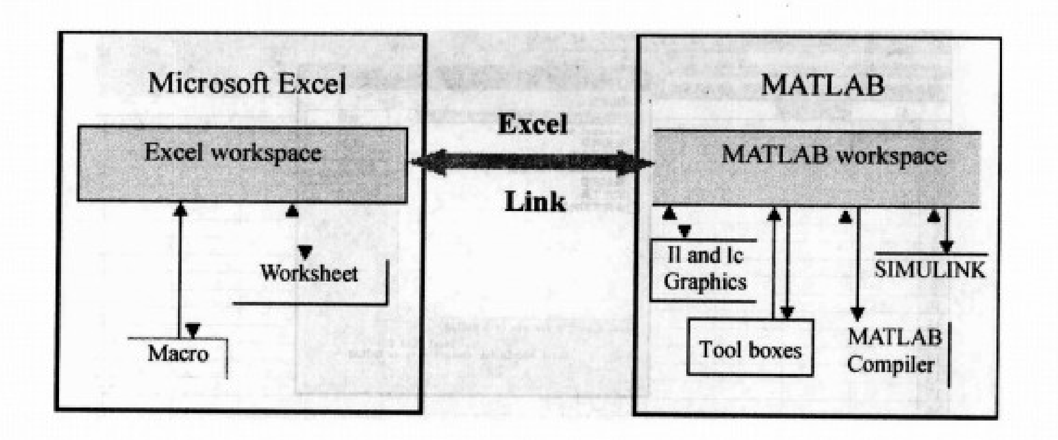


图 3-9

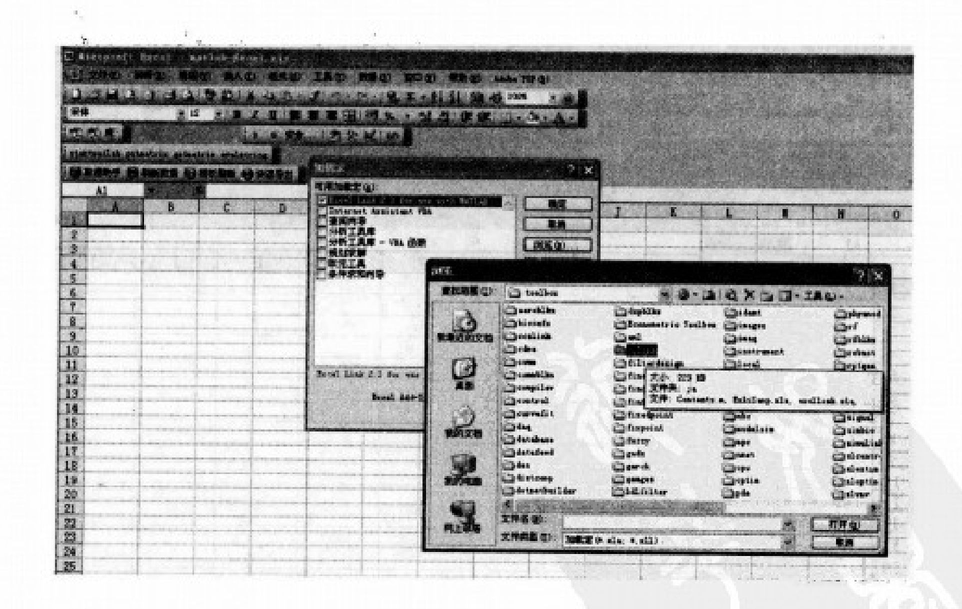


图 3-10

再回到加载宏界面：在 Excel link2. 3 for use with MATLAB 选项上打勾，点击确定，如图 3-11 所示。

如果在 Excel 的左上方出现 startmatlab、putmatrix、getmatrix、evalstring 这样的 Excel-link 则说明加载成功，如图 3-12 所示。

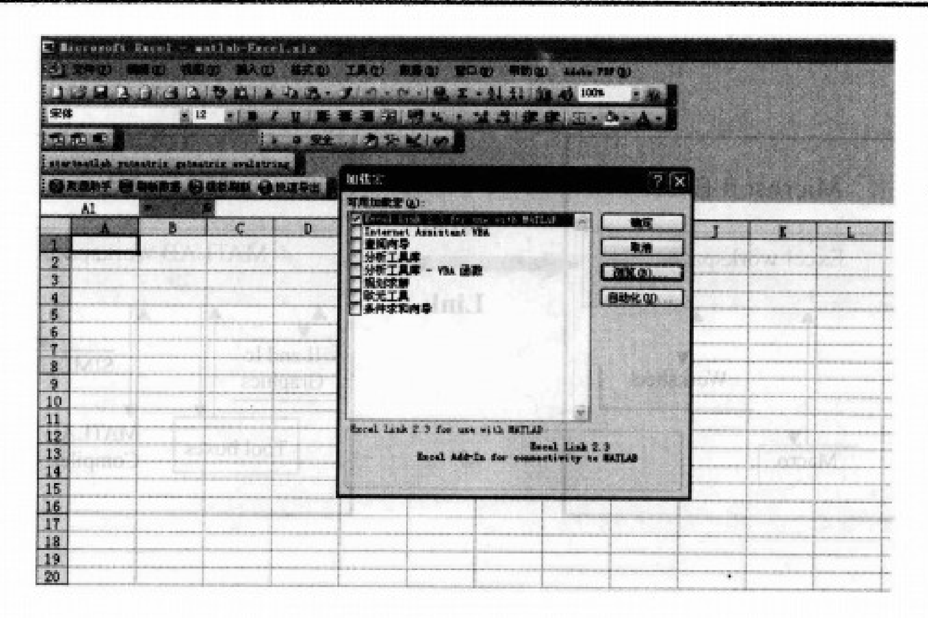


图 3-11

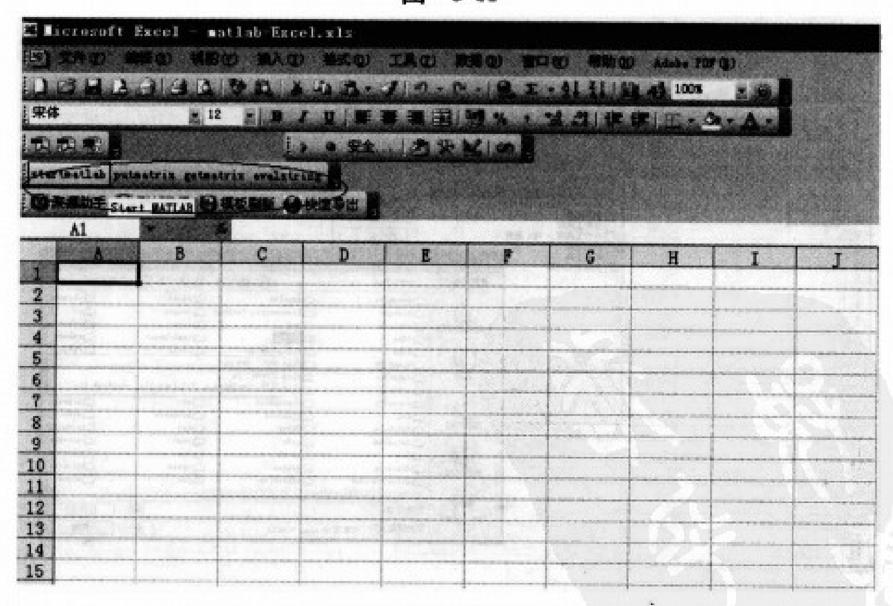


图 3-12

2. Excel-link 使用方法

startmatlab：点击启动 MATLAB。

putmatrix：将 Excel 的数据传输到 MATLAB 中，如图 3-13 所示。

在 MATLAB 中，可以看到传入到 MATLAB 中的矩阵 x，就计算 y = sin(x)，如图 3-14 所示。

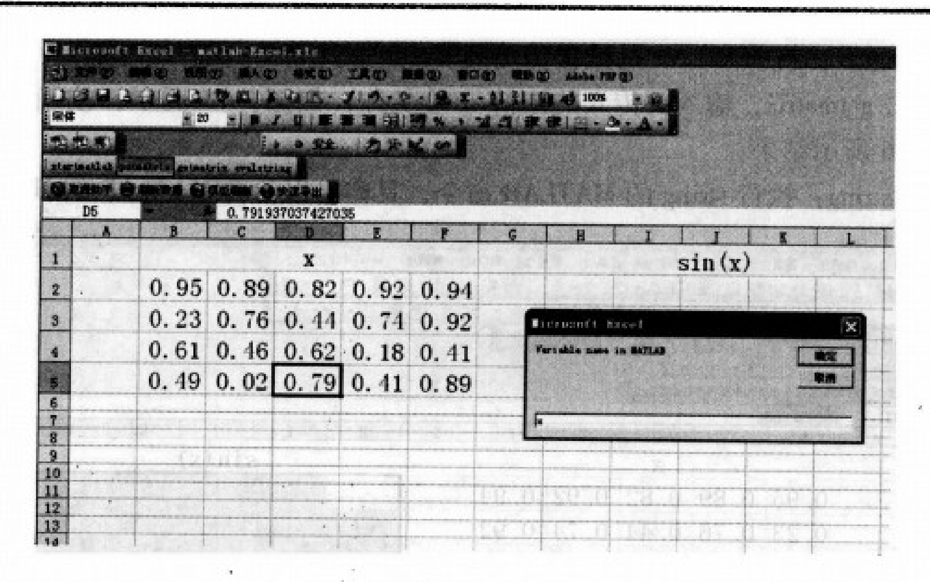

x
sin(x)
0.95 0.89 0.82 0.92 0.94
0.23 0.76 0.44 0.74 0.92
0.61 0.46 0.62 0.18 0.41
0.49 0.02 0.79 0.41 0.89
0.791937037427035

图 3-13

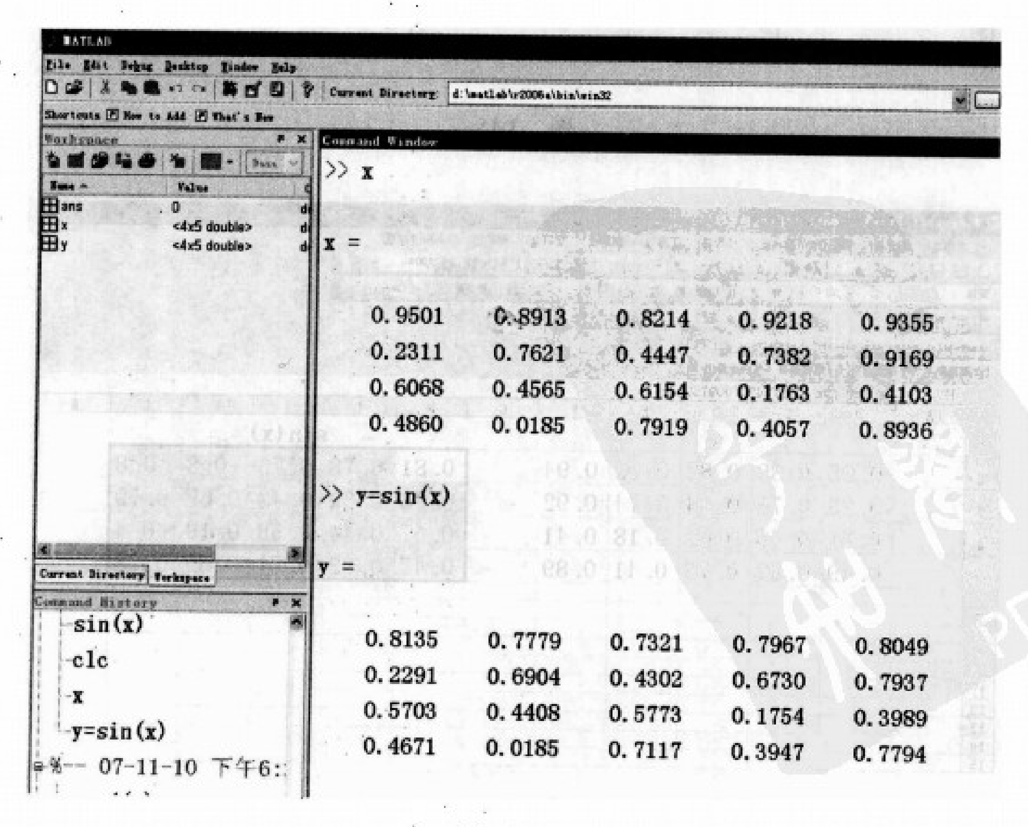

>> x
x =
0.9501 0.8913 0.8214 0.9218 0.9355
0.2311 0.7621 0.4447 0.7382 0.9169
0.6068 0.4565 0.6154 0.1763 0.4103
0.4860 0.0185 0.7919 0.4057 0.8936
>> y=sin(x)
y =
0.8135 0.7779 0.7321 0.7967 0.8049
0.2291 0.6904 0.4302 0.6730 0.7937
0.5703 0.4408 0.5773 0.1754 0.3989
0.4671 0.0185 0.7117 0.3947 0.7794
sin(x)
clc
x
y=sin(x)
%-- 07-11-10 下午6:

图 3-14

getmatrix：将 MATLAB 的数据传输到 Excel 中。

点击 getmatrix，输入要传入的矩阵变量名称，如图 3-15 所示；按确定，得如图 3-16 所示结果。

evalstring：执行 string 的 MATLAB 命令，具体可以参看 MATLAB 的 Help。

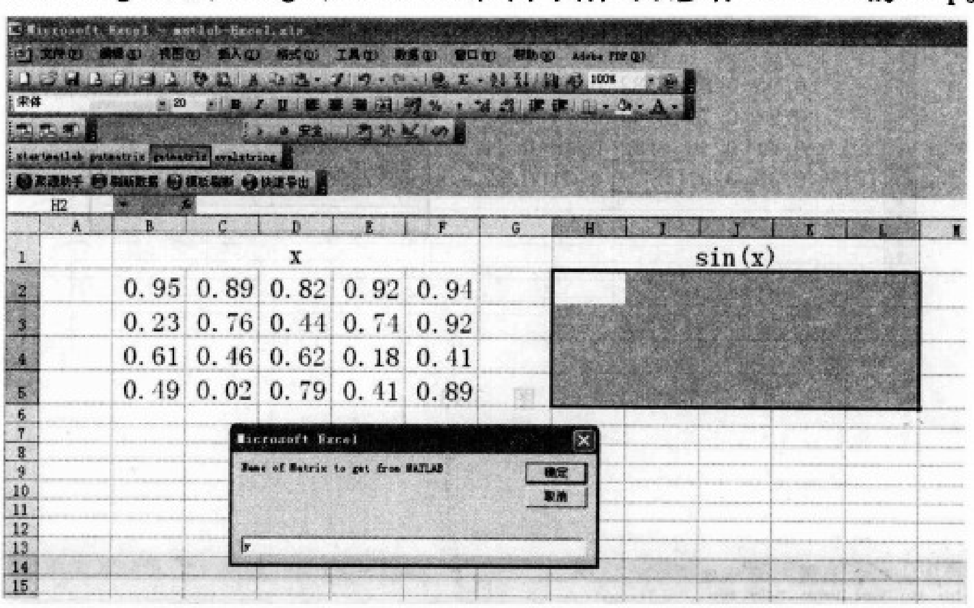


图 3-15

	x						sin(x)				
2	0.95	0.89	0.82	0.92	0.94		0.81	0.78	0.73	0.8	0.8
3	0.23	0.76	0.44	0.74	0.92		0.23	0.69	0.43	0.67	0.79
4	0.61	0.46	0.62	0.18	0.41		0.57	0.44	0.58	0.18	0.4
5	0.49	0.02	0.79	0.41	0.89		0.47	0.02	0.71	0.39	0.78

图 3-16

第 4 章　优化算法的基本结构

4.1　常用的算法搜索结构

优化模型为：

$$(fS)\begin{cases}\min f(\boldsymbol{x})\\ \text{s. t. } \boldsymbol{x}\in S\end{cases}$$

在最优化问题的求解过程中，所用的算法一般是迭代方法。它的基本思想是：给定一初始点 $\boldsymbol{x}^{(0)}\in\boldsymbol{R}^n$，按照某一迭代规则产生一个点列 $\{\boldsymbol{x}^{(k)}\}$，使得当 $\{\boldsymbol{x}^{(k)}\}$ 的最后一个点或其极限点是最优化问题的最优解。介绍算法，具体来说主要是介绍初始点的选择（有时是任意的）、迭代点的产生过程以及停止规则等。本节就算法的一般概念、性质、构造途径等作简要介绍。

4.1.1　收敛性的概念

由于迭代算法是以产生一系列的迭代点为目的的，因此算法的收敛性表现在产生的点列 $\{\boldsymbol{x}^{(k)}\}$ 上。

理想的收敛性概念一般是指：设 $\boldsymbol{x}^*$ 为问题 (fS) 的 g. opt，当 $\{\boldsymbol{x}^{(k)}\}$ 是有穷点列时，$\boldsymbol{x}^*$ 为 $\{\boldsymbol{x}^{(k)}\}$ 的最后一个点；当 $\{\boldsymbol{x}^{(k)}\}$ 是无穷点列时，$\boldsymbol{x}^{(k)}\neq\boldsymbol{x}^*$，$\forall k$，但满足 $\lim\limits_{k\to\infty}\boldsymbol{x}^{(k)}=\boldsymbol{x}^*$，则称算法收敛到最优解 $\boldsymbol{x}^*$。

但在实际运算中，这样的要求很难达到。因此我们建立如下实用的收敛性概念。

定义集合 Ω，称之为解集，若算法产生的点列 $\{\boldsymbol{x}^{(k)}\}$ 满足下列条件之一，则称算法收敛：

（1）$\{\boldsymbol{x}^{(k)}\}\cap\Omega\neq\varnothing$。

（2）$\{\boldsymbol{x}^{(k)}\}$ 的任意收敛子列的极限点属于 Ω。

解集 Ω 一般是指具有某种可接受性质的点集。例如：

（1）$\Omega=\{\boldsymbol{x}^*\mid\boldsymbol{x}^*$ 为问题 (fS) 的 g. opt. $\}$，此即理想收敛。

（2）$\Omega=\{\boldsymbol{x}^*\mid\boldsymbol{x}^*$ 为问题 (fS) 的 l. opt. $\}$。

（3）$\Omega=\{\boldsymbol{x}^*\mid\boldsymbol{x}^*$ 满足某种最优性条件$\}$。

（4）$\Omega=\{\boldsymbol{x}^*\mid\boldsymbol{x}^*\in S$ 且 $f(\boldsymbol{x}^*)\leqslant\beta\}$，其中 β 为可接受的目标函数值的上界。

在求解问题时，初始点的影响是否存在同算法收敛性密切相关，因此有以下概念：

（1）全局收敛性。若算法对任意初始点或任意可行的初始点都收敛，则称算法具有全局收敛性。

（2）局部收敛性。若算法只有当限制初始点在解集 Ω 附近（Ω 非连通时，指在 Ω 某点附近）时才收敛，则称算法具有局部收敛性。

4.1.2 收敛准则（停止条件）

对应于不同的解集定义，可以规定相应的停机条件。如解集 Ω 的定义（3）和（4）本身就可作为停机条件。此外，下列几种停机条件是最常用的，设 $\{x^{(k)}\}$ 为算法产生的点列，$\varepsilon>0$ 为给定的误差限（ε 取值依具体问题而定）：

（1）$\|x^{(k+m)}-x^{(k)}\|<\varepsilon$（一般取 $m=1$）。

（2）$\dfrac{\|x^{(k+1)}-x^{(k)}\|}{\|x^{(k)}\|}<\varepsilon$。

（3）$|f(x^{(k+1)})-f(x)^{(k)}|<\varepsilon$。

在具体求解问题时，应对算法及问题进行必要的分析，从而选择合适的停机条件；否则，可能出现停机但没有得到解的情况，我们称之为早停。

4.1.3 收敛速度

设 $\{x^{(k)}\}$ 为算法产生的点列，$\{x^{(k)}\}$ 收敛于 x^*，且 $x^{(k)}\neq x^*$，$\forall k$

$$\lim_{k\to\infty}\frac{\|x^{(k+1)}-x^*\|}{\|x^{(k)}-x^*\|}=\beta$$

若 $0<\beta<1$，则称 $\{x^{(k)}\}$ 为线性收敛；若 $\beta=0$，则称 $\{x^{(k)}\}$ 为超线性收敛；若 $\beta=1$，则称 $\{x^{(k)}\}$ 为次线性收敛。

设 $\{x^{(k)}\}$ 收敛于 x^*，且 $x^{(k)}\neq x^*$，$\forall k$，若对某个实数 $p\geq 1$，有

$$\lim_{k\to\infty}\frac{\|x^{(k+1)}-x^*\|}{\|x^{(k)}-x^*\|^p}=\beta,\ 0<\beta<\infty$$

则称 $\{x^{(k)}\}$ 为 p 阶收敛的。

在优化问题的算法中，最常涉及的收敛性有：线性收敛、超线性收敛、二阶收敛。

定理 4-1 设点列 $\{x^{(k)}\}$，$x^{(k)}\neq x^*$，$\forall k$

（1）若 $\exists\alpha\in(0,1)$，使当 k 充分大时，$\dfrac{\|x^{(k+1)}-x^*\|}{\|x^{(k)}-x^*\|}\leqslant\alpha$，则 $\{x^{(k)}\}$ 收敛于 x^*，且至少是线性收敛。

（2）若存在正数列 $\{\alpha_k\} \to 0$，使当 k 充分大时，$\dfrac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|} \leqslant \alpha_k$，则 $\{x^{(k)}\}$ 超线性收敛于 x^*。

（3）$\{x^{(k)}\}$ 超线性收敛于 $x^* \Leftrightarrow \lim\limits_{k\to\infty} \dfrac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|} = 0$。

（4）$\{x^{(k)}\}$ 超线性收敛于 $x^* \Rightarrow \lim\limits_{k\to\infty} \dfrac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)} - x^*\|} = 1$。

定理 4-1（4）表明，若算法超线性收敛，当 k 充分大时，可保证 $\|x^{(k)} - x^*\|$ 同 $\|x^{(k+1)} - x^{(k)}\|$ 是相当的。因此对于超线性收敛的算法，用停机条件 $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ 是合理的。

我们构造算法时，希望其具有超线性收敛速度，至少应有线性收敛速度。

4.1.4　线性搜索算法

在多变量数学规划问题的算法中，多采用线性搜索的模式，即在每一步迭代中进行如下工作（设得到迭代点 $x^{(k)}$）：

（1）确定搜索方向 $d^{(k)}$。

（2）求 λ_k，使 $f(x^{(k)} + \lambda_k d^{(k)}) = \min\{f(x^{(k)} + \lambda d^{(k)}) \mid \lambda \in \mathbf{R}_k\}$。

（3）新迭代点：令 $x^{(k+1)} = x^{(k)} + \lambda_k d^{(k)}$。

此模式中，（1）的 $d^{(k)}$ 的产生方法不同，就得到不同的算法；（2）是表示从 $x^{(k)}$ 点出发沿方向 $d^{(k)}$ 寻找在对步长因子 λ 的一定限制范围内的最小值点，称为线性搜索或一维搜索，$\mathbf{R}_k$ 是针对问题得到的限制集合。

线性搜索涉及的是单变量的最优化问题，一般不能经有限步得到解。理论上，若 $\mathbf{R}_k = \mathbf{R}$，则在一定条件下线性搜索的结果应当有：

$$\left.\frac{\mathrm{d}f(x^{(k)} + \lambda_k d^{(k)})}{\mathrm{d}\lambda}\right|_{\lambda=\lambda_k} = 0, \text{即} [\nabla f(x^{(k)} + \lambda_k d^{(k)})]^{\mathrm{T}} d^{(k)} = 0$$

线性搜索与算法的收敛性及收敛速度有着密切的联系。一般说来，线性搜索达到相应方向上的极小是收敛性所要求的，即精确一维搜索。当 $x^{(k+1)}$ 距离 $x^{(k)}$ 很远时，要求精确一维搜索的必要性不大，而往往需要较多的运算工作量，这时按照一定的要求来选择 λ_k，使得函数下降一定的量，可望得到整体较快的收敛。这类线性搜索称为不精确一维搜索。

4.1.5　二次模型

大多数最优化方法是从二次模型导出的。在讨论算法时，人们常常考虑算法用于二次函数时的特征，在求非线性极小化的问题时，常讨论方法对正定二次函

数的有利特性，并把它作为对算法的一个衡量准则。考虑较多的是二次终结性。

利用二次函数进行讨论有如下优点：

(1) 正定二次函数是容易确定极小的、最简单的光滑函数。

(2) 一般的光滑函数在其极小点 x^* 附近可用正定二次函数很好地逼近。因此，建立在二次模型基础上的方法，当迭代点接近 x^* 时，可望有较快的收敛速度。

(3) 在给定的精度下，用二次函数逼近比用线性函数逼近可在较大的区域内有效。

如果一个算法用于正定二次函数求极小时，能经过有限步迭代达到极小点 x^*，则称此算法具有二次终结性。

许多好的算法都有二次终结性，但并不是说二次终结性是算法好坏的决定性标准。实际上，有不少成功的算法不具有二次终结性，也有些算法虽然有二次终结性但效果不理想。尽管如此，许多好的算法还是同时具有超线性收敛速度和二次终结性。对一般函数而言，具有二次终结性的算法可望在接近极小点时具有很好的收敛性。因此，二次终结性仍可作为选择算法的一个因素。

对于无约束极小化问题，即 (fS) 问题中 $S=\mathbf{R}^n$ 的情况，二次终结性可形象地表示为：

$$\text{二次终结性}=\text{共轭方向}+\text{精确一维搜索}$$

4.1.6　下降算法模型

考虑问题

$$(fS)\begin{cases}\min f(\boldsymbol{x})\\ \text{s.t. } \boldsymbol{x}\in S\end{cases}$$

下降算法的思想是：使算法产生点列 $\{\boldsymbol{x}^{(k)}\}$ 对应的函数列 $\{f(\boldsymbol{x}^k)\}$ 为严格单调下降数列，即 $\forall k, f(\boldsymbol{x}^{(k+1)})<f(\boldsymbol{x}^{(k)})$。

依照线性搜索模式，首先要找下降可行方向。

设 $\bar{\boldsymbol{x}}\in S$，对方向 $\boldsymbol{d}$，若 $\exists\delta>0$，$(\delta=\delta(\bar{\boldsymbol{x}},\boldsymbol{d}))$，使 $f(\boldsymbol{x}+\delta\boldsymbol{d})<f(\bar{\boldsymbol{x}})$，$\forall\lambda\in(0,\delta)$，则称 $\boldsymbol{d}$ 为 $f(\boldsymbol{x})$ 在 $\bar{\boldsymbol{x}}$ 点的下降方向。

易证，若 $f(\boldsymbol{x})$ 在 $\bar{\boldsymbol{x}}$ 点可微，$\boldsymbol{d}$ 使 $\nabla f(\bar{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{d}<0$，则 $\boldsymbol{d}$ 为下降方向。

对问题 (fS)，只找到下降方向，有时还不能产生新的迭代点，还要求为可行方向。即 $\exists\delta>0$，使 $\bar{\boldsymbol{x}}+\lambda\boldsymbol{d}\in S$，$\forall\lambda\in(0,\delta)$。

同时满足下降方向与可行方向条件的方向称为下降可行方向。

4.2　一维搜索算法

求解最优化问题的基本方法是给定一个初始可行点 $x^{(0)} \in D$，由这个初始可行点出发，依次产生一个可行点列 $x^{(1)}$，$x^{(2)}$，…，$x^{(k)}$，…，记为 $\{x^{(k)}\}$，使得或者某个 $x^{(k)}$ 恰好是问题的最优解，或者该点列 $\{x^{(k)}\}$ 收敛到问题的一个最优解 x^*。这就是我们平时所说的迭代算法。在迭代算法中，由点 $x^{(k)}$ 迭代到 $x^{(k+1)}$ 时，要求 $f(x^{(k+1)}) \leqslant f(x^{(k)})$，这种算法称为下降算法。点列 $\{x^{(k)}\}$ 的产生常采取两步来完成：首先，在可行域内 $x^{(k)}$ 点处求一个方向 $p^{(k)}$，使得 $f(x)$ 沿方向 $p^{(k)}$ 移动时方向有所下降，一般称这个方向为下降方向或搜索方向；其次，以 $x^{(k)}$ 为出发点，以 $p^{(k)}$ 为方向作射线，$x^{(k+1)} = x^{(k)} + \alpha p^{(k)}$，其中 $\alpha > 0$。在此射线上求一点 $x^{(k+1)}$，$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$，使得 $f(x^{(k+1)}) \leqslant f(x^{(k)})$，其中 α_k 称为步长。

一维搜索算法是根据可行点 $x^{(k)}$ 与下降方向 $p^{(k)}$，结合 $f(x)$，确定步长 α_k 的算法。经典优化算法一般都是由搜索方向与搜索步长两个迭代步骤组成的。一维搜索算法效果的好坏，直接决定整个优化算法的效果，所以说搜索算法对于优化算法是至关重要的。一维搜索算法根据其搜索方法的特点主要分为精确一维搜索算法与非精确一维搜索方法。下面主要介绍三种具有代表性的搜索算法，分别是精确一维搜索算法的进退法、黄金分割法，非精确一维搜索算法的沃尔夫（Wolfe）法。

测试函数为：

$$F(x_1,x_2) = x_1^3 + x_2^2 - 10x_1x_2 + 1$$

MATLAB 程序表达式为：

```
function f = objfun(x)
f = x(1)^3 + x(2)^2 - 10 * x(1) * x(2) + 1;
```

初始点(0,0)，搜索方向(1,1)

图 4-1 的 MATLAL 程序为：

```
x = 0:0.1:3;
y = x;
[X,Y] = meshgrid(x,y);
Z = X.^3 + Y.^2 - 10 * X. * Y + 1;
surfc(X,Y,Z)
```

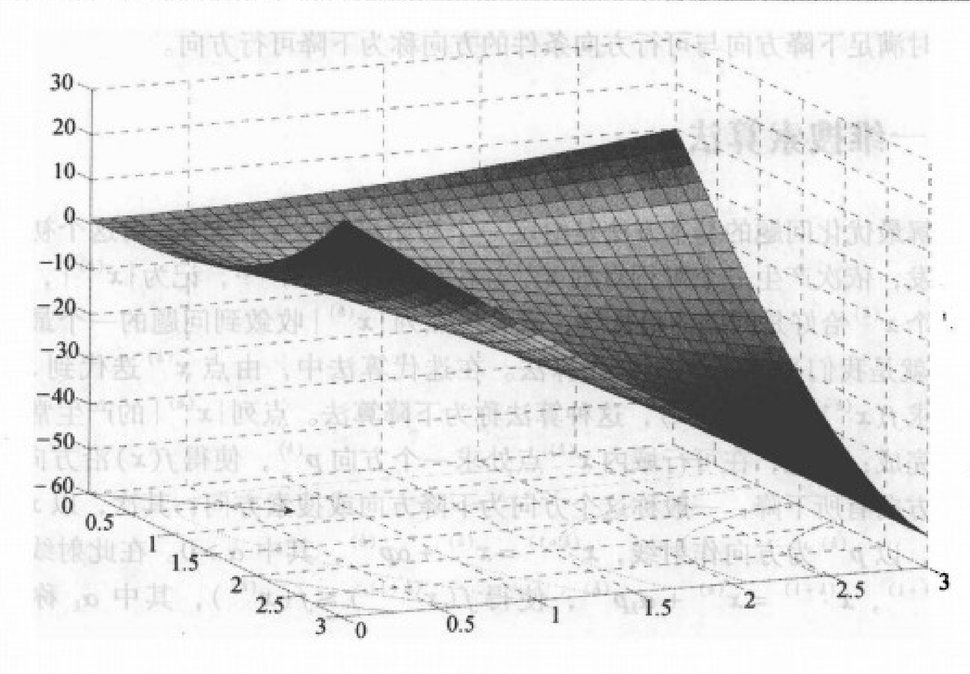


图 4-1

4.2.1 黄金分割法（精确一维搜索）

黄金分割法是一种常用的精确一维搜索。如果设 $f(x)$ 在初始区间 $[a, b]$ 上是一个下单峰函数，则 $f(x)$ 在 $[a, b]$ 有唯一的极小点 x^*，黄金分割法就是求解这样极小点的一种有效方法。

黄金分割法的算法步骤为：

给定 a，$b(a<b)$，控制误差 $\varepsilon>0$。

一般取 $[a, b]$ 为 $[0, 1]$，当然也可以根据需要调整。如果目标函数对变量非常灵敏，可以将步长的可行范围缩小，以提高搜索精度；反之，可以将步长的可行范围放大，以提高算法的计算速度。

步骤1：令 $x_2=a+0.618(b-a)$，$f_2=f(x_2)$ 转步骤2。

步骤2：令 $x_1=a+0.382(b-a)$，$f_1=f(x_2)$ 转步骤3。

步骤3：若 $|b-a|\leqslant\varepsilon$，则 $x^*=\dfrac{a+b}{2}$，停；否则转步骤4。

步骤4：若 $f_1<f_2$，则 $b=x_2$，$x_2=x_1$，$f_2=f_1$，转步骤2；

若 $f_1=f_2$，则 $a=x_1$，$b=x_2$，转步骤1；

若 $f_1>f_2$，则 $a=x_1$，$x_1=x_2$，$f_2=f_1$，转步骤5；

步骤5：令 $x_2=a+0.618(b-a)$，$f_2=f(x_2)$，转步骤3。

MATLAB 程序源码为：

```
function Pa = golden(x,p,LowBound,UpBound)
% code by ariszheng
% email:ariszheng@gmail.com
% golden is function for golden linear search
% x(k+1) = x(k) + Pa * p;
% input:
%   x:搜索初始点
%   p:搜索方向
%   LowBound,UpBound:搜索区间
% output:
%   Pa:搜索步长
% ===========golden linesearch ======================
% disp('it is the golden linereasch');
%   搜索步长范围[0,1]可以根据需要修改
    gold_a = LowBound;
    gold_b = UpBound;
    z2 = gold_a +0.618 * (gold_b - gold_a);%% step1
    f2 = objfun(x + z2 * p);
    z1 = gold_a +0.382 * (gold_b - gold_a);%% step2
    f1 = objfun(x + z1 * p);
    in_itera =0;
    while abs (gold_b - gold_a) > 0.0000000001   % abs a = 0, b = 1 golden
linesearch
        in_itera = in_itera + 1;
        %f1,f2
        if f1 <f2
            gold_b = z2;
            z2 = z1;
            f2 = f1;
            z1 = gold_a +0.382 * (gold_b - gold_a);
            f1 = objfun(x + z1 * p);
            continue
        elseif f1 == f2
              gold_a = z1;
```

```
            gold_b = z2;
            z2 = gold_a + 0.618 * (gold_b - gold_a); %% step1
            f2 = objfun(x + z2 * p);
            z1 = gold_a + 0.382 * (gold_b - gold_a); %% step2
            f1 = objfun(x + z1 * p);
            continue
        else
            gold_a = z1;
            z1 = z2;
            f1 = f2;
            z2 = gold_a + 0.618 * (gold_b - gold_a);
            f2 = objfun(x + z2 * p);
        end
    end
    Pa = (gold_a + gold_b)/2;   % Pa   x(k+1) = x(k) + Pa * p;
    % Pa, in_itera
    % obj(x + Pa * p)
    % ================== golden linesearch end ============
```

MATLAB 函数的使用方法：

1. 定义目标函数：(M 文件)

```
function f = objfun(x)
f = x(1)^3 + x(2)^2 - 10 * x(1) * x(2) + 1;
```

2. 确定初始搜索点与搜索方向

```
X = [0,0]; P = [1,1]
```

3. 调用函数计算

以[0,1]为搜索区间

```
Pa = golden(X,P,0,1)
Pa = 1
```

4.2.2 进退法

对于精确一维搜索，一般都要求$f(x)$在初始区间 $[a, b]$ 上是一个下单峰函数。进退法就是求解函数$f(x)$下单峰区间 $[a, b]$ 的一种常用方法。

进退法的算法步骤为：

给定初始点 $\boldsymbol{x}_0$，初始步长 $\Delta x(>0)$。

步骤 1：计算 $f(x_0)$，转步骤 2。

步骤 2：$x_1 = x_0 + \Delta x$，计算 $f(x_1)$。

若 $f(x_1) \leqslant f(x_0)$，则转步骤 3；否则转步骤 5。

步骤 3：令 $\Delta x = 2\Delta x$，$x_2 = x_1 + \Delta x$，计算 $f(x_2)$。

若 $f(x_1) \leqslant f(x_2)$，则得到区间 $[x_0, x_2]$ 为初始区间，停；

若 $f(x_1) > f(x_2)$，则转步骤 4。

步骤 4：令 $x_0 = x_1, x_1 = x_2, f(x_0) = f(x_1), f(x_1) = f(x_2)$，转步骤 3。

步骤 5：令 $\Delta x = 2\Delta x$，$x_2 = x_0 - \Delta x$，计算 $f(x_2)$。

若 $f(x_0) \leqslant f(x_2)$，则得到区间 $[x_2, x_1]$ 为初始区间，停；

若 $f(x_0) > f(x_2)$，则转步骤 6。

步骤 6：令 $x_1 = x_0, x_0 = x_2, f(x_1) = f(x_0), f(x_0) = f(x_2)$，转步骤 5。

程序的流程图如图 4-2 所示。

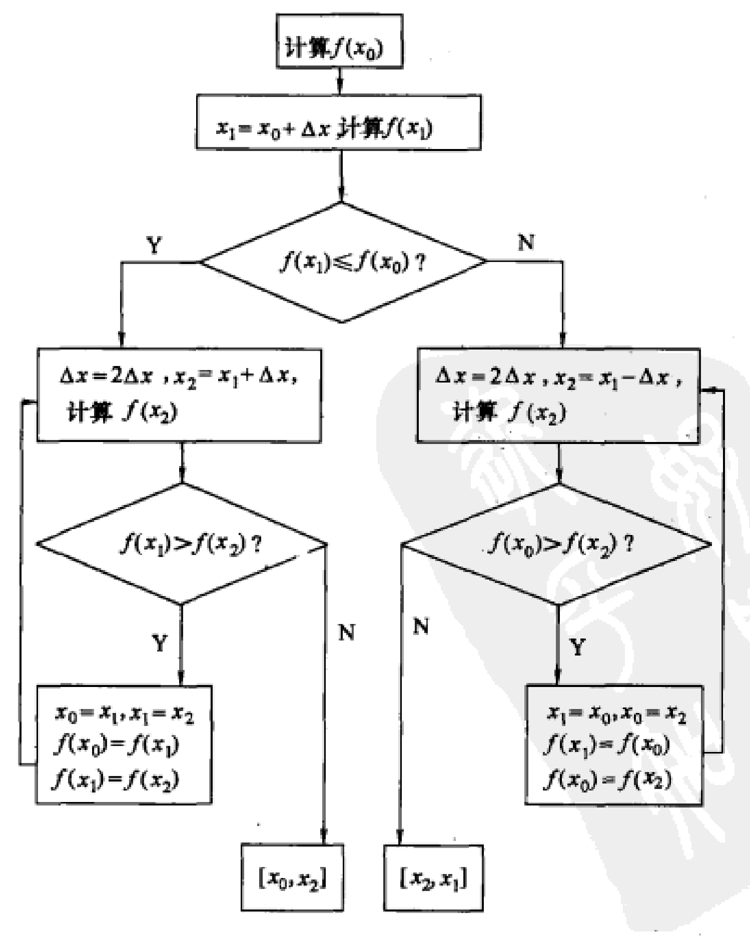


图 4-2

MATLAB 的源码为：

```
function[LowBound,UpBound] = TryBound(x0,step0,StartOpint,SearchDirection)
% code by ariszheng
% email:ariszheng@gmail.com
% TryBound is function to find fit Bounds for precision search
% input:
%    x0:探测初始点
%    step0:初始探测步长
% output:
%    [LowBound,UpBound]:可行搜索区间
% ========== TryBound Start ==================
step = step0;
%% 步骤 1
f0 = TryObjfun(x0,StartOpint,SearchDirection);
%% 步骤 2
x1 = x0 + step0;
f1 = TryObjfun(x1,StartOpint,SearchDirection);
if f1 <= f0
%% 步骤 3
    while true
        step = 2 * step;
        x2 = x1 + step;
        f2 = TryObjfun(x2,StartOpint,SearchDirection);
        if f1 <= f2
            LowBound = x0;
            UpBound = x2;
            break;
        else
%% 步骤 4
            x0 = x1;
            x1 = x2;
            f0 = f1;
            f1 = f2;
        end
```

```
        end
else
%%步骤5
        while true
            step =2 * step;
            x2 = x0 - step;
            f2 = TryObjfun(x2,StartOpint,SearchDirection);
            if f0  <=f2
                LowBound = x2;
                UpBound = x1;
                break;
            else
                %%步骤6
                x1 = x0;
                x0 = x2;
                f1 = f0;
                f0 = f2;
                end
            end
end
% ========== TryBound End ==================
```

MATLAB 函数的使用方法为:

1. 定义目标函数:(M-文件)

```
function f = objfun(x)
f = x(1)^3 + x(2)^2 - 10 * x(1) * x(2) + 1;
```

2. 确定初始搜索点与搜索方向

```
X = [0,0];P = [1,1]
```

3. 定义一维搜索函数

```
function f = TryObjfun(a,StartOpint,SearchDirection)
f = objfun(StartOpint + a. * SearchDirection);
    %计算 x = StartOpint + a. * SearchDirection 的函数值
    % StartOpint = [0,0]
    % SearchDirection = [1,1]
```

4. 确定初始搜索区间

```
% 以0为初始探测点,0.01为探测步长。
    >> [LowBound,UpBound] = TryBound(0,0.01,[0,0],[1,1])
LowBound = 2.5500
UpBound = 10.2300
```

图4-3为步长可行区间的(步长—函数值)图像。

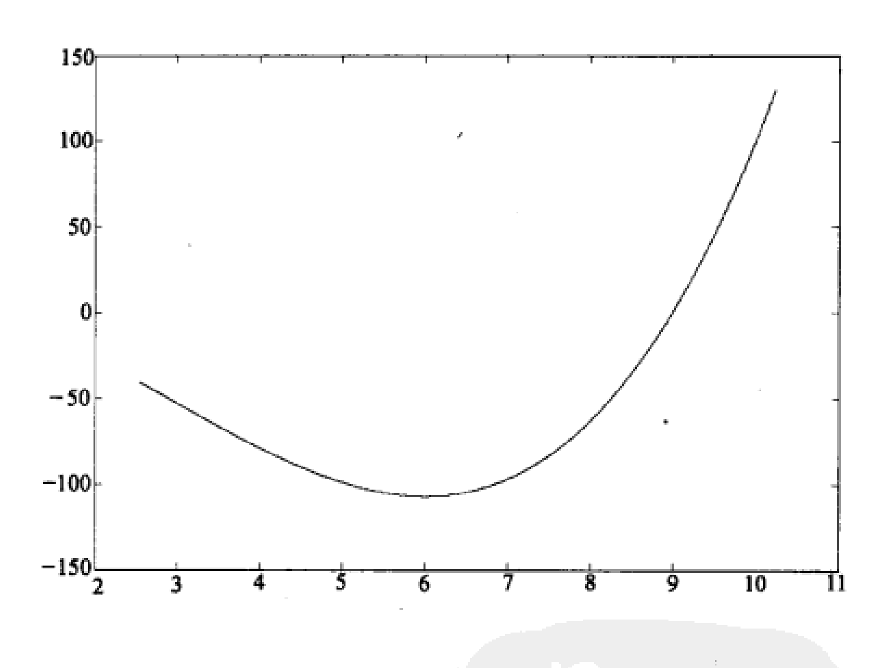


图 4-3

5. 根据下单峰的初始区间用精确一维方法确定下降步长

```
    >> Pa = golden([0,0],[1,1],LowBound,UpBound)
Pa = 6.0000
    >> TryObjfun(6,[0,0],[1,1])
    ans = -107
```

4.2.3 沃尔夫法

在解非线性规划问题时，一维搜索一般很难达到真正的精确值。为了达到比较高的精度，往往需要计算很多个函数，计算量大。为此引入了不精确一维搜索，它只计算少量的函数值就得以得到一个下降点。沃尔夫（Wolfe）算法就是一种非常有效的不精确一维搜索算法。

设 $f(x)$ 可微，取 $\mu \in \left(0, \frac{1}{2}\right), \sigma \in (\mu, 1)$，选取 $\alpha_k > 0$ 使其满足：

(1) $f(\boldsymbol{x}^{(k)}) - f(\boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}) \geqslant -\mu \alpha_k \boldsymbol{g}^{(k)\mathrm{T}} \boldsymbol{p}^{(k)}$

(2) $|\nabla f_{\mathrm{T}}(\boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}) \boldsymbol{p}^{(k)}| \leqslant -\sigma \boldsymbol{g}^{(k)\mathrm{T}} \boldsymbol{p}^{(k)}$

得到一个单调下降的收敛点列 $\{\alpha_k\}$。

算法的几何描述如图 4-4 所示。

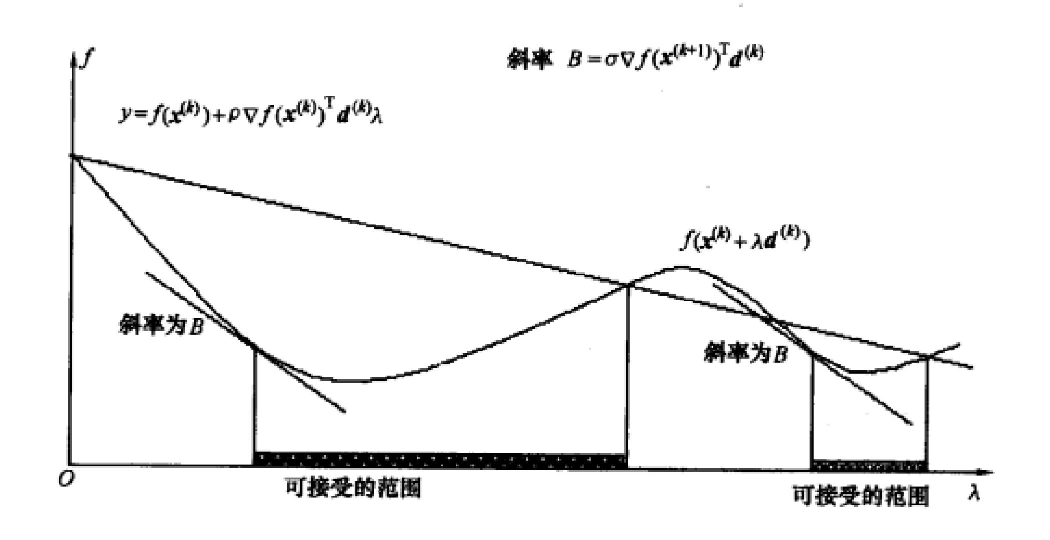


图 4-4

沃尔夫算法的具体步骤如下：

步骤 1：给定 $\mu \in \left(0, \frac{1}{2}\right), \sigma \in (\mu, 1)$，令 $a = 0$，$b = +\infty$，$\alpha = 1$，$j = 0$。

步骤 2：$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$，计算 $f(\boldsymbol{x}^{(k+1)}), g(\boldsymbol{x}^{(k+1)})$。

若 α 满足（1）和（2），则 $\alpha_k = \alpha$，停。

若 α 不满足（1），令 $j = j + 1$，转步骤 3。

若 α 不满足（2），令 $j = j + 1$，转步骤 4。

步骤 3：令 $b = \alpha$，$\alpha = \dfrac{\alpha + a}{2}$，转步骤 2。

步骤 4：令 $a = \alpha$，$\alpha = \min\left\{2\alpha, \dfrac{\alpha + b}{2}\right\}$，转步骤 2。

MATLAB 程序的源码为：

```
function    Pa = wolfe(x,p)
```

```
% code by ariszheng
% email:ariszheng@gmail.com
% wolfe is function for wolfe linear search
% x(k+1)=x(k)+Pa*p;
% input:
%    x:搜索初始点
%    p:搜索方向
% output:
%    Pa:搜索步长
% ========== golden linesearch ====================
% initial the parament 设置参数(可以根据需要调整)
wolfe_u=0.1;
wolfe_l=0.5;
wolfe_a=0;
wolfe_b=1000;
wolfe_Pa=1;% 初始步长
MaxIterNum=50;% 搜索最大迭代次数
j=0;
N=length(x);
%%%%% start the iteration%%%%%%%%%
x0=x;
f0=objfun(x0);
g1=diffobjfun(x0);
for i=1:MaxIterNum
    x=x0+wolfe_Pa*p;
    f1=objfun(x);
    g2=diffobjfun(x);
    % condition1 判别条件一
    temp1=f0-f1+wolfe_u*wolfe_Pa*g1*p';
    % condition 2 判别条件二
    temp2=abs(g2*p')+wolfe_l*g1*p';
    if(temp1>=0&&temp2<=0)
    %% 如果同时符合条件一、二迭代停止
        Pa=wolfe_Pa;
```

```
        break
    elseif temp1 <0
    %%条件一不符合
        j=j+1;
        wolfe_b=wolfe_Pa;
        %计算新步长
        wolfe_Pa=(wolfe_Pa+wolfe_a)/2;
    else
    %%条件二不符合
        j=j+1;
        wolfe_a=wolfe_Pa;
        %计算新步长
        wolfe_Pa=min(2*wolfe_Pa,(wolfe_Pa+wolfe_b)/2);
    end
end
Pa=wolfe_Pa;
in_itera=j;%总计算次数
% ================== wolfe linesearch end ===========
```

MATLAB 函数的使用方法为:

1. 定义目标函数与目标函数导数:(M 文件)

```
function f=objfun(x)
f=x(1)^3+x(2)^2-10*x(1)*x(2)+1;
function df=diffobjfun(x)
%f=x(1)^3+x(2)^2-10*x(1)*x(2)+1;
df(1)=3*x(1)^2-10*x(2);
df(2)=2*x(2)-10*x(1);
```

计算导数的方法有多种,目标函数简单并可以直接求出导数;当然,也可以使用数值差分的方法计算导数。

2. 确定初始搜索点与搜索方向

```
X=[0,0];P=[1,1]
```

3. 调用函数计算

```
Pa=wolfe(X,P)
Pa=9
```

4.3 MATLAB 函数 Fminbnd

单变量固定区间的最优模型为：

$$\min_{x} f(x), \text{ s. t: } x_1 < x < x_2$$

Fminbnd 函数是 MATLAB 中最强大的在固定区间上的一维搜索函数，采用的是黄金分割（Golden Section Search）与抛物插值法（Parabolic Interpolation）相结合的算法，因此不需要设定初始迭代点。

Fminbnd 函数的语法为：

```
x = fminbnd(fun,x1,x2)
x = fminbnd(fun,x1,x2,options)
[x,fval] = fminbnd(...)
[x,fval,exitflag] = fminbnd(...)
[x,fval,exitflag,output] = fminbnd(...)
```

函数输入：

Fun：目标函数

x1：可行区间下界

x2：可行区间上界

options：函数参数

函数输出：

x：最优点

fval：最优点对应的函数值

exitflag：函数结束信息

output：函数计算信息

MATLAB 函数的使用方法为：

1. 定义目标函数：(M 文件 myfun. m)

以 $f(x) = \sin(x) + e^x$ 为目标函数对应的 M 文件：

```
function f = myfun(x)
f = sin(x) + exp(x);
```

2. 调用函数 Fminbnd

OPTIONS = optimset('display','iter')；设置优化参数，显示迭代过程

OPTIONS 设置参看附录 Optimization option

[x,fval,exitflag,output] = fminbnd(@ myfun,0,5,options)；求解目标函数在[0,5]区间上的最小点及其对应的函数值

计算结果（具体迭代步骤）：

Func-count	x	f（x）	Procedure
1	1.90983	7.69502	initial
2	3.09017	22.0322	golden
3	1.18034	4.18022	golden
4	0.917271	3.2964	parabolic
5	0.566905	2.29983	golden
6	0.350366	1.76283	golden
7	0.216538	1.45662	golden
8	0.133828	1.27663	golden
9	0.0827103	1.16884	golden
10	0.0511178	1.10354	golden
11	0.0315925	1.06368	golden
12	0.0195253	1.03924	golden
13	0.0120673	1.02421	golden
14	0.00745798	1.01494	golden
15	0.00460929	1.00923	golden
16	0.0028487	1.0057	golden
17	0.00176059	1.00352	golden
18	0.00108811	1.00218	golden
19	0.000672486	1.00135	golden
20	0.000415619	1.00083	golden
21	0.000256867	1.00051	golden
22	0.000158752	1.00032	golden
23	9.81144e-005	1.0002	golden
24	6.0638e-005	1.00012	golden

最优性终止：当前 x 满足 OPTIONS. TOLX 的终止条件 1.000000e-004。

x = 6.0638e −005　最优点

fval = 1.0001　最优点对应的函数值

exitflag =　1

output =

　　iterations：23

　　funcCount：24

　　algorithm：' golden section search，parabolic interpolation '

　　message：[1x112 char]

第5章　线性规划

线形规划（Linear Programming）是运筹学中应用最广泛的模型之一。由于其理论与方法研究比较成熟，许多问题常借助线形规划模型来求解，而且线形规划为某些非线性规划问题的解法起到间接作用。

5.1　线性规划的模型结构

$$
\begin{aligned}
&\max(\min)\ f = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\text{s.t.}\quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leqslant (=, \geqslant)\, b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leqslant (=, \geqslant)\, b_2 \\
& \qquad \vdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leqslant (=, \geqslant)\, b_m \\
& x_1, x_2, \cdots, x_n \geqslant 0
\end{aligned}
$$

线性规划的标准形式为：

$$
\begin{aligned}
&\max f = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\text{s.t.}\quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
& \qquad \vdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\
& x_1, x_2, \cdots, x_n \geqslant 0
\end{aligned}
$$

转化成矩阵的表示形式为：

$$
\begin{aligned}
&\max f(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\
\text{s.t.}\quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
& \boldsymbol{x} \geqslant \boldsymbol{0}
\end{aligned}
$$

标准形式的可行域是凸集合（凸集合概念可以参看相关章节）。

各种形式的线性规划都可以化为标准形式。

（1）若给出的线性规划是极大化目标函数，则有

$$\min f(\boldsymbol{x}) = \max[-f(\boldsymbol{x})]$$

（2）若第 i 个约束为不等式，则可增加松弛变量，把原约束化为等式约束。

若 $\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i$，则加入 $x_{n+1} \geqslant 0$，得到：

$$\sum_{j=1}^{n} a_{ij}x_j + x_{n+1} = b_i$$

目标函数保持不变，即松弛变量 x_{n+1} 的目标函数系数为 $c_{n+1}=0$。

类似地，若第 i 个约束为：

$$\sum_{j=1}^{n} a_{ij}x_j \geqslant b_i$$

则

$$\sum_{j=1}^{n} a_{ij}x_j - x_{n+1} = b_i, x_{n+1} \geqslant 0$$

（3）若第 i 个等式约束中 $b_i<0$，则用 -1 乘该等式两端。

（4）若第 j 个变量 x_j 没有非负限制，此时称 x_j 为自由变量，则引入两非负变量，即 x_j'，$x_j''\geqslant 0$，令 $x_j = x_j' - x_j''$，将其代入目标函数以及约束中去。

例如：

$$\begin{aligned} \min f = & -x_1 - x_2 - x_3 \\ \text{s. t.} \quad & 7x_1 + 3x_2 + 9x_3 \leqslant 1 \\ & 8x_1 + 5x_2 + 4x_3 \leqslant 1 \\ & 6x_1 + 9x_2 + 5x_3 \leqslant 1 \\ & x_1, x_2, x_3 \geqslant 0 \end{aligned}$$

引入松弛变量 x_4，x_5，$x_6\geqslant 0$，原式变为线性规划的标准形式为：

$$\begin{aligned} \max f = & x_1 + x_2 + x_3 + 0x_4 + 0x_5 + 0x_6 \\ \text{s. t.} \quad & 7x_1 + 3x_2 + 9x_3 + x_4 + 0x_5 + 0x_6 = 1 \\ & 8x_1 + 5x_2 + 4x_3 + 0x_4 + 1x_5 + 0x_6 = 1 \\ & 6x_1 + 9x_2 + 5x_3 + 0x_4 + 0x_5 + x_6 = 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geqslant 0 \end{aligned}$$

5.2　线性规划的单纯形法

单纯形法（Simplex Method）是求解线性规划问题的通用方法。单纯形是美国数学家 G. B. 丹齐克于 1947 年首先提出来的。它的理论根据是：线性规划问题的可行域是 n 维向量空间 R^n 中的多面凸集，其最优值如果存在，必在该凸集的某顶点处达到。顶点所对应的可行解称为基本可行解。单纯形法的基本思想是：先找出一个基本可行解，对它进行鉴别，看是否是最优解；若不是，则按照一定法则转换到另一改进的基本可行解，再鉴别；若仍不是，则再转换，按此重复进行。因基本可行解的个数有限，故经有限次转换必能得出问

题的最优解。如果问题无最优解也可用此法判别。单纯形法的一般解题步骤可归纳如下：①把线性规划问题的约束方程组表示成标准型方程组，找出基本可行解作为初始基本可行解；②若基本可行解不存在，即约束条件有矛盾，则问题无解；③若基本可行解存在，从初始基本可行解作为起点，根据最优性条件和可行性条件，引入非基变量取代某一基变量，找出目标函数值更优的另一基本可行解；④按步骤 3 进行迭代，直到对应检验数满足最优性条件（这时目标函数值不能再改善），即得到问题的最优解；⑤若迭代过程中发现问题的目标函数值无界，则终止迭代。

数学优化中，由 George Dantzig 发明的单纯形法是线性规划问题数值求解的流行技术。有一个算法与此无关，但名称类似，它是 Nelder-Mead 法或称下山单纯形法，由 Nelder 和 Mead 发现（1965 年），这是用于优化多维无约束问题的一种数值方法，属于更一般的搜索算法的类别。这二者都使用了单纯形的概念，它是 N 维中的 $N+1$ 个顶点的凸包，是一个多胞体：直线上的一个线段，平面上的一个三角形，三维空间中的一个四面体，等等。

5.2.1 单纯形算法

$$
\begin{aligned}
&\max f(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & & \min g(\boldsymbol{x}) = -\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}\\
&\text{s.t.}\quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} & \Rightarrow\quad & \text{s.t}\quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\\
&\qquad\ \ \boldsymbol{x} \geqslant \boldsymbol{0} & & \qquad \boldsymbol{x} \geqslant \boldsymbol{0}
\end{aligned}
$$

设矩阵 $\boldsymbol{A}$ 的秩为 m。

算法说明：

本书以线性函数最大化（即 max）给出，但是由于 MATLAB 的内置函数都是以最小化形式进行编程的，为了编写的程序与 MATLAB 的风格一致，本书程序是以最小化的标准形式进行编程实现的。两种形式在算法步骤上只有目标函数系数向量 $\boldsymbol{c}$ 的正负号相反。请读者在阅读时注意两者的区别。

步骤 1：取得一个初始可行基 $\boldsymbol{B}$，写出初始基可行解 $\boldsymbol{x} = \begin{pmatrix}\boldsymbol{x}_B\\ \boldsymbol{x}_N\end{pmatrix} = \begin{pmatrix}\boldsymbol{B}^{-1}\boldsymbol{b}\\ \boldsymbol{0}\end{pmatrix}$，以及当前的目标函数值 $z = \boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{x}_B = \boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{B}^{-1}\boldsymbol{b}$，计算所有检验数 σ_j，$j=1, 2, \cdots, n$，$\boldsymbol{\sigma}_N = \boldsymbol{c}_N^{\mathrm{T}} - \boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{B}^{-1}\boldsymbol{N}$。

步骤 2：考察所有检验数 σ_j，$j=1, 2, \cdots, n$，若所有检验数 $\sigma_j \geqslant 0$，则当前基为最优解，停。否则转步骤 3。

步骤 3：令 $\sigma_k = \max\{\sigma_j | \sigma_j > 0\}$。若 $\boldsymbol{B}^{-1}\boldsymbol{p}_k \leqslant 0$，则无最优解，停。否则转步骤 4。

步骤4：令 $\theta=\min\left\{\frac{(\boldsymbol{B}^{-1}\boldsymbol{b})_i}{(\boldsymbol{B}^{-1}\boldsymbol{p}_k)_i}\middle|(\boldsymbol{B}^{-1}\boldsymbol{p}_k)_i>0\right\}=\frac{(\boldsymbol{B}^{-1}\boldsymbol{b})_r}{(\boldsymbol{B}^{-1}\boldsymbol{p}_k)_r}$，用 x_k 代替 x_r 得新基。

步骤5：新得的基可行解及判别数：

$$\forall j=1,2,\cdots,n$$

$$a_{rj}=\frac{a_{rj}}{a_{rk}},$$

$$a_{ij}=a_{ij}-\frac{a_{rj}}{a_{rk}}a_{ik},i\neq r,$$

$$b_r=\frac{b_r}{a_{rk}},$$

$$b_i=b_i-\frac{b_r}{a_{rk}}a_{ik},i\neq r,$$

$$\sigma_j=\sigma_j-\frac{a_{rj}}{a_{rk}}\sigma_k,$$

转步骤2。

5.2.2 单纯形表格法的 MATLAB 程序：simplexTab

MATLAB 优化库函数都是以最小化为标准，所以 simplexTab（mat，numFreeVar）程序也以最优化为标准。

simplexTab 的使用方法为：

先将一般的线性规划变为线性规划的标准形式，再构建初始单纯形表格，输入程序。例如：

$$\begin{cases}\max f=-x_1-x_2-x_3\\ \text{s.t.}\quad 7x_1+3x_2+9x_3\leqslant 1\\ \qquad 8x_1+5x_2+4x_3\leqslant 1\\ \qquad 6x_1+9x_2+5x_3\leqslant 1\\ \qquad x_1,x_2,x_3\geqslant 0\end{cases}\Rightarrow\begin{cases}\min g=x_1+x_2+x_3+0x_4+0x_5+0x_6\\ \text{s.t.}\quad 7x_1+3x_2+9x_3+x_4+0x_5+0x_6=1\\ \qquad 8x_1+5x_2+4x_3+0x_4+1x_5+0x_6=1\\ \qquad 6x_1+9x_2+5x_3+0x_4+0x_5+x_6=1\\ \qquad x_1,x_2,x_3,x_4,x_5,x_6\geqslant 0\end{cases}$$

单纯形表格为：

1. 初始输入表

	x_1	x_2	x_3	x_4	x_5	x_6	b	theta
	7	3	9	1	0	0	1	
A	8	5	4	0	1	0	1	
	6	9	5	0	0	1	1	
系数	-1	-1	-1	0	0	0	0(g 值)	

注：(g 值) 处显示标准型（求 min）的目标函数值。

2. 迭代过程表

	x_1	x_2	x_3	x_4	x_5	x_6	b	theta
	7	3	9	1	0	0	1	
A	8	5	4	0	1	0	1	
	6	9	5	0	0	1	1	
判别数	-1	-1	-1	0	0	0	0	

在 MATLAB 的 command window 输入：

```
mat = [7  3  9  1  0  0  1;
       8  5  4  0  1  0  1;
       6  9  5  0  0  1  1;
      -1 -1 -1  0  0  0  0];
numFreeVar = 3;% (注释：自由变量个数)
simplexTab (mat, numFreeVar)
```

在 MATLAB 的 command window 输出：

初始结果：

```
the best Pivot is 2 row and 1 col
the simplex table is

   7.0000    3.0000    9.0000   1.0000        0        0   1.0000   0.1429
   8.0000    5.0000    4.0000        0   1.0000        0   1.0000   0.1250
 (6.0000)    9.0000    5.0000        0        0   1.0000   1.0000   0.1667
  -1.0000   -1.0000   -1.0000        0        0        0        0        0
```

第一次转换结果：

```
the best Pivot is 1 row and 3 col
the simplex table is

 0   -1.3750   (5.5000)   1   -0.8750   0   0.1250   0.0227
 1    0.6250    0.5000    0    0.1250   0   0.1250   0.2500
 0    5.2500    2.0000    0   -0.7500   1   0.2500   0.1250
 0   -0.3750   -0.5000    0    0.1250   0   0.1250        0
```

第二次转换结果：

```
the best Pivot is 1 row and 2 col
the simplex table is
      0  ( -0.2500)  1.0000   0.1818  -0.1591        0   0.0227  -0.0909
 1.0000     0.7500        0  -0.0909   0.2045        0   0.1136   0.1515
      0     5.7500        0  -0.3636  -0.4318   1.0000   0.2045   0.0356
      0    -0.5000        0   0.0909   0.0455        0   0.1364        0
```

第三次转换结果：

```
      0        0   1.0000   0.1660  -0.1779   0.0435   0.0316  -0.0909
 1.0000        0        0  -0.0435   0.2609  -0.1304   0.0870   0.1515
      0   1.0000        0  -0.0632  -0.0751   0.1739   0.0356   0.0356
      0        0        0   0.0593   0.0079   0.0870   0.1542        0
It is the end!
```

It is the end!（表示程序运行完毕）

最后的单纯形表为：

	x_1	x_2	x_3	x_4	x_5	x_6	b	theta
A	0	0	1	0.1660	-0.1779	0.0435	0.0316	-0.0909
	1	0	0	-0.0435	0.2609	-0.1304	0.0870	0.1515
	0	1	0	-0.0632	-0.0751	0.1739	0.0356	0.0356
判别数	0	0	0	0.0593	0.0079	0.0870	0.1542	0

$\boldsymbol{x}=(0.0316,0.0870,0.0356)$, $g=0.1542$, $f=-x_1-x_2-x_3=-0.1542$

主程序：

simplexTab(mat, numFreeVar)

输入单纯形表 mat

自由变量个数 numFreeVar

子程序：由主函数调用

1. **function newMat = interChange(mat, row1, row2)**

单纯形表的两行对调

2. **function newMat = multiFromRowToRow(mat, fromRow, toRow, multiplier)**

行数据非主元素消去处理

3. **function newMat = multMat(mat, row, mult)**

行乘以 mult 得新行数据

4. **function newMat = pivot(mat, row, col)**

以 mat(row, col) 为轴做旋转

```
==========================================================
function simplexTab(mat,numFreeVar)
% Recode by ariszheng
%2007 08 09
%%
%输入:
%mat          单纯形表
%numFreeVar   自由变量个数
format short
%检验和寻找最佳判别数
%maxRow,maxCol 单纯形表的行数与列数
maxRow = length(mat(:,1));
maxCol = length(mat(1,:));
%判别数
objEntryExcludingMaxPayOff = mat(maxRow,1:maxCol - 2);
%寻找最小判别数
[objEnt bestColToPivot] = min(objEntryExcludingMaxPayOff);
%%
%循环迭代
while(objEnt <0)
        %出基可行解列
        lastColExcludingObjEntry = mat(1:(maxRow - 1),maxCol);
        %入基可行解列
        ithColExcludingObjEntry = mat(1:(maxRow - 1),bestColToPivot);
        %找出最小正比数以及其对应的行
        a = lastColExcludingObjEntry./ithColExcludingObjEntry;
        [val bestRowToPivot] = min(a);
        sprintf('the best Pivot is %d row and %d col',bestRowToPivot,bestColTo-
Pivot)
        disp('the simplex table is');
        [mat,[a;0]]
        disp('press any key to continue');
        pause;
        if(val <0)
```

```
            [s indices] = sort(a);
            if(max(a) >0)
                count = 1;
            while(s(count) <0)
                count = count +1;
            end
            bestRowToPivot = indices(count);
          end
        end
        if(length(a) ==0)
            length(a)
            return
        end
        %旋转变换
        mat = pivot(mat,bestRowToPivot,bestColToPivot);
        %寻找最小判别数
        objEntryExcludingMaxPayOff = mat(maxRow,1:maxCol -2);
        [objEnt bestColToPivot] = min(objEntryExcludingMaxPayOff);
        %当所有的判别数都大于0的时候迭代停止
end
%%
sprintf('the best Pivot is %d row and %d col',bestRowToPivot,bestColToPivot)
disp('the simplex table is');
[mat,[a;0]]
disp('It is the end!');
=====================================================
function newMat = interChange(mat,row1, row2)
temp = mat(row1,:);
mat(row1,:) = mat(row2,:);
mat(row2,:) = temp;
newMat = mat;
function newMat = multiFromRowToRow(mat,fromRow,toRow,multiplier)
rG = mat(fromRow,:) * multiplier;
mat(toRow,:) = rG + mat(toRow,:);
```

```
newMat = mat;
function newMat = multMat(mat,row,mult)
% multiply a row of a matrix by a nonzero constant
mat(row,:) = mat(row,:) * mult;
newMat = mat;
function newMat = pivot(mat, row,col)
% normalize this row
mat(row,:) = mat(row,:)./mat(row,col);% make the leading a number a 1
for r = 1:length(mat(:,1))
    if(r ~ = row)
      mat = multFromRowToRow(mat,row,r, -mat(r,col));
    end
end
newMat = mat;
```

5.3 linprog 函数

linprog 函数在 MATLAB 优化工具箱 Optimization-Toolbox 中。

linprog 针对的线性函数模型为

$$\begin{aligned} &\min \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x} \\ &\text{s. t.} \quad \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b} \\ &\qquad \boldsymbol{Aeq}\,\boldsymbol{x} = \boldsymbol{beq} \\ &\qquad \boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub} \end{aligned}$$

这里 $\boldsymbol{f}$，$\boldsymbol{x}$，$\boldsymbol{b}$，$\boldsymbol{beq}$，$\boldsymbol{lb}$，$\boldsymbol{ub}$ 是向量，$\boldsymbol{A}$，$\boldsymbol{Aeq}$ 是矩阵。

Linprog 计算算法为：

1. 约束优化问题的拉格朗日乘法（即：内点法）

有关该算法的介绍本章不讲，可以参看约束规划问题的相关章节。

2. 单纯形法 Simplex

linprog 函数格式为：

1. x = linprog(f,A,b)s

求解目标 min f' * x 约束 A * x <= b.

输入：f：目标函数系数向量

A：不等式约束系数矩阵

b：不等式约束常数向量

输出：x：最优解

2. x = linprog(f,A,b,Aeq,beq)

输入：Aeq：等式约束系数矩阵

Beq：等式约束常数向量

3. x = linprog(f,A,b,Aeq,beq,lb,ub)

输入：lb：x的可行域下界

ub：x的可行域上界

4. x = linprog(f,A,b,Aeq,beq,lb,ub,x0)

输入：x0：初始迭代点（这个与linprog使用的算法有关）

5. x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)

输入：options优化参数设置

6. [x,fval] = linprog (...)

输出：fval：最优目标函数值

7. [x,lambda,exitflag] = linprog(...)

输出：lambda，exitflag：算法停止原因

8. [x,lambda,exitflag,output] = linprog(...)

输出：output：优化结果的约束信息

9. [x,fval,exitflag,output,lambda] = linprog(...)

输出：lambda：结果x对应的拉格朗日乘子

输出参数说明：

Exitflag：返回算法迭代停止原因

返回值：1　算法收敛于解x，即x是线性规划的最优解

0　算法达到最大迭代次数停止迭代，即x不一定是线性规划的最优解

-2　算法没有找到可行解，即算法求解失败，问题的可行解集合为空

-3　原问题无解，即最优解可能为正（负）无穷大

-4　在算法中出现除零问题或其他问题，导致变量中出现非数值情况

-5　线性规划的原问题与对偶问题都不可解

-7　可行搜索方向向量过小，无法再提高最优解质量

Lambda：返回解的拉格朗日乘子与约束符合情况

Lower：求得解越下界

Upper：求得解越上界

Neqlin：求得解不满足不等式约束

Eqlin：求得解不满足等式约束

Output：返回算法信息

Algorithm：计算时使用的优化算法

Cgiterations：共轭梯度迭代次数（只有大规模算法时有）

iterations：算法迭代次数

Exit message：返回结束信息

5.3.1 实例演示 1（对应程序 test2）

$$\min f = -x_1 - x_2 - x_3$$

$$\text{s.t.} \quad 7x_1 + 3x_2 + 9x_3 \leqslant 1$$

$$8x_1 + 5x_2 + 4x_3 \leqslant 1$$

$$6x_1 + 9x_2 + 5x_3 \leqslant 1$$

$$x_1, x_2, x_3 \geqslant 0$$

使用[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb,ub)

Command window 输入

```
f=[-1,-1,-1]  %目标函数系数
A=[7,3,9;8,5,4;6,9,5]:%不等式约束的系数矩阵
b=[1,1,1]  %不等式约束的b
Aeq=[]  %等式约束的系数矩阵(该问题无等式约束 Aeq 为空)
beq=[]  %等式约束的 beq(该问题无等式约束 beq 为空)
lb=[0,0,0]  %变量的下界
ub=[]  %变量的上界(无上界约束,ub 为空)
[x,fval,exitflag,output,lambda]=linprog(f,A,b,Aeq,beq,lb,ub)
```

Command window 输出

```
Optimization terminated.(优化算法计算结束)
x=[0.0870,0.0356,0.0316](最优解)
fval= -0.1542(最优解对应的函数值)
exitflag=1(算法收敛于解 x,即 x 是线性规划的最优解)
output =
    iterations:7(算法迭代 7 次)
    algorithm:'large-scale:interior point'(使用的算法是内点法)
    cgiterations:0(共轭梯度迭代 0 次,没有使用共轭梯度迭代)
    message:'Optimization terminated.'(算法正常停止)
lambda =
```

ineqlin:[3x1 double]
eqlin:[0x1 double]
upper:[3x1 double]
lower:[3x1 double]

lambda. ineqlin = [0.0593,0.0079,0.087](符合约束条件)

使用 linprog 的单纯形法(对应程序 test3)

Command window 输入

```
f=[-1,-1,-1]  %目标函数系数
A=[7,3,9;8,5,4;6,9,5];%不等式约束的系数矩阵
b=[1,1,1]  %不等式约束的b
Aeq=[]  %等式约束的系数矩阵(该问题无等式约束,Aeq为空)
beq=[]  %等式约束的beq(该问题无等式约束,beq为空)
lb=[0,0,0]  %变量的下界
ub=[]  %变量的上界(无上界约束,ub为空)
options=optimset('LargeScale','off','Simplex','on','Display','iter');
(设置优化方法为单纯形法,展示每次迭代的结果)
[x,fval,exitflag,output,lambda]=linprog(f,A,b,Aeq,beq,lb,ub,[],options)
(注释:optimset 参考附录 optimset 的具体说明)
```

Command window 输出

he default starting point is feasible, skipping Phase 1.

Phase 2: Minimize using simplex.

Iter	Objective f' * x	Dual Infeasibility A' * y + z - w - f
0	0	1.73205
1	-0.125	0.625
2	-0.142857	0.357143
3	-0.15415	0

Optimization terminated.

x = [0.0870,0.0356,0.0316]

fval = -0.1542

exitflag = 1

output =

iterations:3(算法迭代3次)

algorithm:'medium scale:simplex'(使用的是中规模的单纯形法)

```
        cgiterations: [ ]
        message: 'Optimization terminated. '
lambda =
        ineqlin: [3x1 double]
        eqlin: [0x1 double]
        upper: [3x1 double]
        lower: [3x1 double]
```

5.3.2 实例演示 2（对应程序 test4）

$$\min f = x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7x_7 + 8x_8 + 9x_9 + 10x_{10}$$

$$\text{s.t.}\quad 7x_1 + 3x_2 + 9x_3 \leq 1$$

$$8x_1 + 5x_2 + 4x_3 \leq 1$$

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} = 1$$

$$x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7,\ x_8,\ x_9,\ x_{10} \geq 0$$

使用以下函数格式：

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb,ub,x0,f,A,b,Aeq,
                                   beq,lb,ub,x0,options)
```

Command window 输入：

```
f = 1:10   %1:10 就是 1,2,...,10 即目标函数系数
A = [7,3,9,0,0,0,0,0,0,0;
     8, 5, 4,0,0,0,0,0,0,0]   %不等式约束的系数矩阵
b = [1,1]   %不等式约束的 b
Aeq = [1,1,1,1,1,1,1,1,1,1]   %等式约束的系数矩阵(该问题无等式约束,Aeq
                                为空)
beq = 1;   %等式约束的 beq(该问题无等式约束,beq 为空)
lb = [0,0,0,0,0,0,0,0,0,0]   %变量的下界
ub = [ ]   %变量的上界(无上界约束,ub 为空)
options = optimset('Display','iter');%展示算法每次迭代结果
x0 = [ ]   %不设置算法初始迭代点
[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

Command window 输出：

```
Residuals:    Primal        Dual          Duality       Total
               Infeas        Infeas        Gap            Rel
               A*x-b         A'*y+z-f      x'*z          Error
```

```
Iter  0: 2.87e+003 2.89e+001 1.55e+004 5.50e+003
Iter  1: 2.16e+002 5.75e-015 1.12e+003 1.24e+002
Iter  2: 1.83e-014 7.32e-015 1.39e+001 1.77e+000
Iter  3: 4.97e-016 9.08e-014 2.12e+000 3.91e-001
Iter  4: 5.78e-015 9.85e-015 2.28e-001 6.06e-002
Iter  5: 3.93e-015 1.44e-014 1.12e-001 3.04e-002
Iter  6: 6.48e-012 2.04e-015 1.44e-003 4.00e-004
Iter  7: 7.02e-016 1.28e-015 1.52e-007 4.23e-008
Iter  8: 3.51e-016 5.05e-016 1.56e-014 4.19e-015
Optimization terminated.
x = [0.0000, 0.2000, 0.0000, 0.8000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000]
fval = 3.6000
exitflag = 1
output =
        iterations: 8
        algorithm: 'large-scale: interior point'
        cgiterations: 0
        message: 'Optimization terminated.'
lambda =
     ineqlin: [2x1 double]
     eqlin: -4.0000
     upper: [10x1 double]
     lower: [10x1 double]
```

第 6 章　无约束优化算法

无约束最优化问题，是指优化问题的可行解集为 $\mathbf{R}^n$，无约束的标准形式为：

$$\min f(x)$$

$$f:\mathbf{R}^n \rightarrow \mathbf{R} \tag{6-1}$$

本章主要介绍无约束问题的基本思想和 MATLAB 相关函数的调用。对无约束问题的基本结构理论的了解，有利于更有效地使用 MATLAB 相关无约束优化函数。无约束算法的种类繁多，本章重点介绍最速下降法、牛顿法与拟牛顿法（变尺度法）。拟牛顿法（变尺度法）是现在公认求解无约束优化问题的最有效的方法。

本章所用 BenchMark 函数为 Banana function：

$$f(x) = 100[x(2) - x(1)^2]^2 + [1 - x(1)]^2 \tag{6-2}$$

Banana function 的最优点为[1, 1]，函数对应的最小值为 0。

函数的 MATLAB 表达式为：

```
function f = BanaFun(x)
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
```

函数图像在[-2, 2] × [-1, -3]上的图像为三维图，如图 6-1 所示。

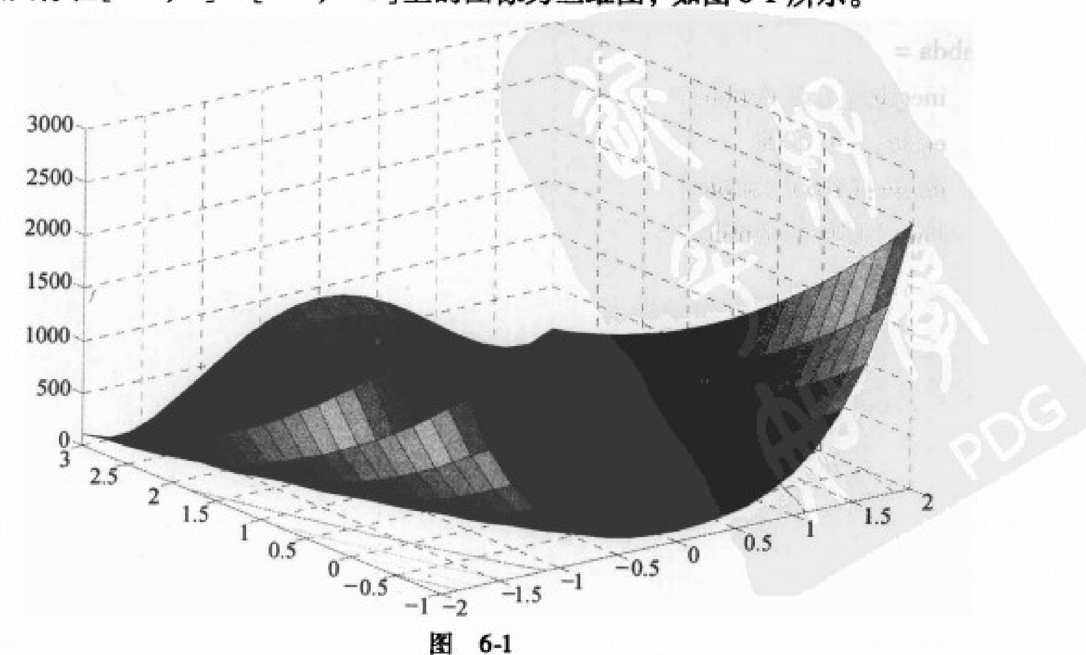


图　6-1

图像用的 MATLAB 程序为：

```
axis off;
x = -2:.2:2;
y = -1:.2:3;
[xx,yy] = meshgrid(x,y);
zz = 100 * (yy - xx.^2).^2 + (1 - xx).^2;
surfc(xx,yy,zz);(三维图)
```

或者

```
surface(x,y,zz,'EdgeColor',[.8 .8 .8]);(俯视图)
```

6.1　最优性条件

1. 极小点的一阶必要条件

设$f(\boldsymbol{x}):\mathbf{R}^n \to \mathbf{R}$为连续可微函数，如果$\boldsymbol{x}^*$为局部极小点，则$\boldsymbol{x}^*$为驻点，即梯度 $\nabla f(\boldsymbol{x}^*)=0$。

2. 极小点的二阶必要条件

设$f(\boldsymbol{x}):\mathbf{R}^n \to \mathbf{R}$为二次连续可微函数，如果$\boldsymbol{x}^*$为局部极小点，则$\boldsymbol{x}^*$为驻点，即梯度 $\nabla f(\boldsymbol{x}^*)=0$，二阶 Hesse 阵 $\nabla^2 f(\boldsymbol{x}^*)$半正定。

3. 极小点的二阶充分条件

设$f(\boldsymbol{x}):\mathbf{R}^n \to \mathbf{R}$为二次连续可微函数，如果梯度 $\nabla f(\boldsymbol{x}^*)=0$，二阶 Hesse 阵 $\nabla^2 f(\boldsymbol{x}^*)$正定，则$\boldsymbol{x}^*$为$f$的局部极小点，$\boldsymbol{x}^* \in R^n$。

以上三个定理为搜索最优点以及判断一个点是否为最优点的基本依据。经典的优化算法的停止条件为 $\nabla f(\boldsymbol{x}^*)=0$，例如在程序中$\|\nabla f(\boldsymbol{x}^*)\| \leqslant 1\times 10^{-8}$，即在导数范数小于某特定误差限时停止。误差限较大，则算法迭代次数减少，计算时间缩短，但是解的质量降低；误差限较小，则算法迭代次数增加，计算时间增长，但是解的质量提高；误差限一般为1×10^{-8}，可以根据实际情况设定合适的误差限。当然，还有极小点的二阶必要与极小点的二阶充分条件，对 $\nabla^2 f(\boldsymbol{x}^*)$的判定，由于目标函数比较复杂，二阶导数矩阵的计算量极大，所以一般算法都在迭代过程中对 $\nabla^2 f(x^{(k)})$进行修正，得到 $\nabla^2 f(x^{(k+1)})$，在修正过程中始终保持 $\nabla^2 f(\boldsymbol{x}^*)$的正定性，以此方法解决极小点的二阶条件问题。

6.2 最速下降法

6.2.1 算法原理

最速下降法是早期的优化算法，其理论根据函数的一阶泰勒展开：

由$f(\boldsymbol{x}+\alpha\boldsymbol{d})=f(\boldsymbol{x})+\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+o(\alpha\|\boldsymbol{d}\|)$

得到$f(\boldsymbol{x}+\alpha\boldsymbol{d})-f(\boldsymbol{x})=\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+o(\alpha\|\boldsymbol{d}\|)$

根据下降要求$f(\boldsymbol{x}+\alpha\boldsymbol{d})-f(\boldsymbol{x})\leqslant 0$

故$\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+o(\alpha\|\boldsymbol{d}\|)\leqslant 0$

实际中要求$\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}\leqslant 0$

根据上式选取合适的d，得$\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}\leqslant 0$。最速下降法取$d=-\nabla f(\boldsymbol{x})$。由于近似地有：

$$f(\boldsymbol{x}+\alpha\boldsymbol{d})-f(\boldsymbol{x})=\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$$

取$\boldsymbol{d}=-\nabla f(\boldsymbol{x})$，则

$$f(\boldsymbol{x}+\alpha\boldsymbol{d})-f(\boldsymbol{x})=-\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\nabla f(\boldsymbol{x})$$

由$\nabla f(\boldsymbol{x})^{\mathrm{T}}\nabla f(\boldsymbol{x})\geqslant 0$知：

$$-\alpha\nabla f(\boldsymbol{x})^{\mathrm{T}}\nabla f(\boldsymbol{x})\leqslant 0$$

最速下降法有全局收敛性，并且是线性收敛的，算法比较简单。一般来说，在实际计算中，最速下降法在开始迭代时效果较好，有时能很快地达到最优解的附近，但是当继续迭代时，常常发生扭摆现象，以致不能达到最优解，如图 6-2 所示。

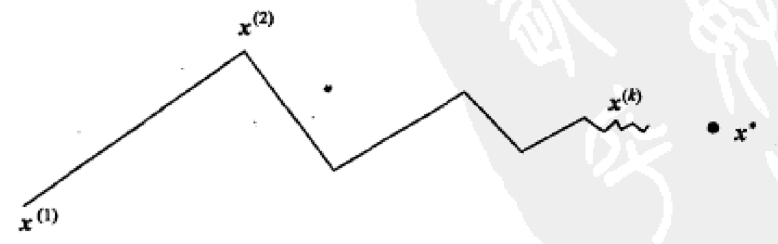


图 6-2

6.2.2 算法步骤

给定控制误差$\varepsilon>0$。

步骤 1：取初始点$\boldsymbol{x}^{(0)}$，令$k=0$。

步骤 2：计算$\boldsymbol{g}^{(k)}=\nabla f(x^{(k)})$。

步骤 3：若$\|\boldsymbol{g}^{(k)}\|\leqslant\varepsilon$，则$\boldsymbol{x}^*=x^{(k)}$，停止计算；否则，令$\boldsymbol{p}^{(k)}=-\boldsymbol{g}^{(k)}$，由一

维搜索步长 α_k，使得

$$f(\boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}) = \min_{\alpha \geq 0} f(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{p}^{(k)})$$

步骤 4：令 $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$，$k = k+1$，转步骤 2。

MATLAB 实现最速下降法的函数为 fminunc，fminunc 函数是 MATLAB 求解无约束优化问题的主要函数，最速下降法仅仅是 fminunc 函数所使用的算法之一。

语法：

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
[x,fval] = fminunc(...)
[x,fval,exitflag] = fminunc(...)
[x,fval,exitflag,output] = fminunc(...)
[x,fval,exitflag,output,grad] = fminunc(...)
[x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

输入参数：

Fun：目标函数，一般用 M 文件形式给出

X0：优化算法初始迭代点

Options：参数设置

函数输出：

X：最优点输出（或最后迭代点）

Fval：最优点（或最后迭代点）对应的函数值

Exitflag：函数结束信息（具体参见 MATLAB Help）

Output：函数基本信息，包括迭代次数、目标函数最大计算次数、使用的算法名称、计算规模

Grad：最优点（或最后迭代点）的导数

Hessian：最优点（或最后迭代点）的二阶导数

6.2.3　程序示例

MATLAB 函数的使用方法：

1. 目标函数程序 BanaFun.m 与 BanaFunWithGrad.m

```
function f = BanaFun(x) （不含导数解析式）
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
function [f,g] = BanaFunWithGrad(x) （含导数解析式）
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
g = [100 * (4 * x(1)^3 - 4 * x(1) * x(2)) + 2 * x(1) - 2; 100 * (2 * x(2) - 2
```

*x(1)^2)];

2. 参数设置(steepdesc.m)

OPTIONS = optimset('LargeScale','off','HessUpdate','steepdesc','gradobj','on','MaxFunEvals',250,'display','iter');

具体参看附录 Optimization option 参数设置。

'LargeScale','off'	大规模计算模式　关闭
'HessUpdate','steepdesc'	Hess 阵修正方式，采用最速下降法(不需要修正)
'gradobj','on'	目标函数导数解析式，on 使用，off 不使用(差分导数)
'MaxFunEvals',250	最大目标函数计算次数，250 次
'display','iter'	显示迭代过程

3. 函数计算(steepdesc.m)

x = [-1.9,2]；初始迭代点

[x,fval,exitflag,output] = fminunc(@BanaFunWithGrad,x,OPTIONS)

计算结果：

迭代次数 Iteration	函数计算次数 Func-count	函数值 f(x)	步长 Step-size	一阶导数最优性 First-order optimality
0	1	267.62		1.23e+003
1	2	214.416	0.000813405	519
2	9	5.83639	0.00098493	13.4
3	15	5.78292	0.000567305	1.58
4	18	5.73127	0.0387979	12.9
5	24	5.68023	0.000583736	1.59
6	27	5.63081	0.0367599	12.5
7	33	5.5819	0.000600291	1.6
8	36	5.53444	0.0349482	12.1
9	42	5.48743	0.000617009	1.61
10	45	5.44173	0.0333245	11.7
11	51	5.39641	0.000633925	1.62
12	54	5.35228	0.0318587	11.3
13	60	5.30849	0.000651074	1.63
⋮	⋮	⋮	⋮	⋮
50	225	4.01396	0.0175609	7.3

51	231	3.98427	0.00106284	1.8
52	234	3.95495	0.0171458	7.16
53	240	3.92561	0.00108886	1.8
54	243	3.89441	0.0181085	7.29
55	249	3.86321	0.00111764	1.82

Maximum number of function evaluations exceeded；达到最大的函数值计算次数。
increase options. MaxFunEvals　建议增加最大的函数值计算次数。
x = −0.9634　　0.9372
fval = 3.8632
exitflag = 0　（Number of iterations exceeded options. MaxIter or number of function evaluations exceeded options. FunEvals.）
output =
　　iterations：56
　　funcCount：250
　　stepsize：0.0011
　　firstorderopt：1.8155
　　algorithm：'medium-scale：Quasi-Newton line search'
　　message：[1x78 char]

从结果可见，在 250 次最大的目标函数计算次数内，算法没有到最优点[1，1]。

将最大目标函数计算次数调高到 2500 次，则

```
OPTIONS =
optimset('LargeScale','off','HessUpdate','steepdesc','gradobj','on','MaxFunEvals',
2500);
x=[-1.9,2];
[x,fval,exitflag,output] = fminunc(@BanaFunWithGrad,x,OPTIONS)
```

计算结果：

Maximum number of iterations exceeded；达到最大迭代次数
increase options. MaxIter. 建议增加最大迭代次数
x =
　0.9770　　0.9542
fval = 5.3808e−004
exitflag = 0

```
output =
        iterations: 401
        funcCount: 1462
        stepsize: 0.0142
        firstorderopt: 0.0574
        algorithm: 'medium-scale: Quasi-Newton line search'
        message: [1x67 char]
```

从以上两个算法的结果可以看出，最速下降法对 bana 函数不是十分有效。

当目标函数不可微或者导数求解复杂时，可以无需提供目标函数的导函数的解析形式，MATLAB 可以使用差分的方法求导（下降方向），对应的代码为：

```
OPTIONS = optimset('LargeScale','off','HessUpdate','steepdesc','MaxFunEvals',250);
x = [-1.9,2];
[x,fval,exitflag,output] = fminunc(@BanaFun,x,OPTIONS)
```

计算结果：

```
Maximum number of function evaluations exceeded;
increase options.MaxFunEvals
x =    -1.2553     1.5640
fval =      5.1002
exitflag =      0
output =
        iterations: 19
        funcCount: 252
        stepsize: 0.0282
        firstorderopt: 10.4149
        algorithm: 'medium-scale: Quasi-Newton line search'
        message: [1x78 char]
```

从计算结果上可以明显看出，对于最速下降法，有导数解析式会比没有导数解析式时，计算质量会有所提高。

6.3 牛顿算法

6.3.1 算法原理

牛顿算法的基本思想是利用二次函数近似目标函数，比最速下降法的一次函

数更进了一步，将次二次函数的极小点作为新的迭代点。

设$f(\boldsymbol{x})$二次连续可微，求解无约束问题

$$\min f(\boldsymbol{x})$$
$$f:\mathbf{R}^n\rightarrow\mathbf{R}$$

若已求得解$\boldsymbol{x}^*$的一个近似点$\boldsymbol{x}^{(k)}$，对$f(\boldsymbol{x}^{(k)}+\boldsymbol{s})$的泰勒展开式取前三项，得到

$$q^{(k)}(\boldsymbol{s})=f(\boldsymbol{x}^{(k)})+\nabla f(\boldsymbol{x}^{(k)})^{\mathrm{T}}\boldsymbol{s}+\frac{1}{2}\boldsymbol{s}^{\mathrm{T}}\ \nabla^2 f(\boldsymbol{x}^{(k)})\boldsymbol{s}$$

其中，$\boldsymbol{s}=\boldsymbol{x}-\boldsymbol{x}^{(k)}$，$\boldsymbol{q}^{(k)}(\boldsymbol{s})$为$f(\boldsymbol{x})$在$\boldsymbol{x}^{(k)}$点附近的二次近似。

设$\nabla^2 f(\boldsymbol{x}^{(k)})$正定，$q^{(k)}(\boldsymbol{x})$有唯一极小点，也是驻点，使得

$$\nabla\boldsymbol{q}^{(k)}(x)=\nabla f(\boldsymbol{x}^{(k)})+\nabla^2 f(\boldsymbol{x}^{(k)})\boldsymbol{s}=0$$

得
$$s^{(k)}=-[\nabla^2 f(\boldsymbol{x}^{(k)})]^{-1}\nabla f(\boldsymbol{x}^{(k)})$$

取
$$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+\boldsymbol{s}^{(k)}\text{或}\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}-[\nabla^2 f(\boldsymbol{x}^{(k)})]^{-1}\nabla f(\boldsymbol{x}^{(k)})$$

由于$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}-[\nabla^2 f(\boldsymbol{x}^{(k)})]^{-1}\nabla f(\boldsymbol{x}^{(k)})$中求矩阵逆计算复杂，可以通过解方程

$$\nabla^2 f(\boldsymbol{x}^{(k)})\boldsymbol{s}^{(k)}=-\nabla f(\boldsymbol{x}^{(k)})$$

求解$\boldsymbol{s}^{(k)}$。

6.3.2　算法步骤

给定控制误差$\varepsilon>0$。

步骤1：取初始点$\boldsymbol{x}^{(0)}$，令$k=0$。

步骤2：计算$\nabla f(\boldsymbol{x}^{(k)})$，$\nabla^2 f(\boldsymbol{x}^{(k)})$。

解$\nabla^2 f(\boldsymbol{x}^{(k)})\boldsymbol{s}^{(k)}=-\nabla f(\boldsymbol{x}^{(k)})$

得解$\boldsymbol{s}^{(k)}$，$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+\boldsymbol{s}^{(k)}$。

步骤3：若$\|\boldsymbol{s}^{(k)}\|<\varepsilon$，则停机，$\boldsymbol{x}^*=\boldsymbol{x}^{(k+1)}$；否则$k=k+1$，转步骤2。

牛顿算法收敛速度为二阶，是其最大的优点。牛顿算法对正定二次函数一步迭代即达最优解，因此具有二次终结性。

6.3.3　算法特点

牛顿算法有一些致命的缺点，常导致方法进行中产生困难，甚至失败。

（1）牛顿算法是局部收敛的。当初始点选择不当时，往往导致算法不收敛。

（2）牛顿算法不是下降算法，当二阶 Hesse 阵非正定时，不能保证产生的方向是下降方向。

（3）二阶 Hesse 阵$\nabla^2 f(x^k)$必须可逆，否则算法进行困难。

(4) 对函数要求苛刻（二阶可微，Hesse 阵可拟），运算量巨大。

由于牛顿算法的收敛速度快，人们想尽办法去克服牛顿算法的上述致命缺点。方法有：

(1) 减小计算量的直接修正。

(2) 带一维搜索的牛顿算法。

(3) Goldstein-price（1967）方法。

(4) 强迫 Hesse 矩阵正定的方法。

在这个改进牛顿算法的过程中，又产生了各种各样的有效算法，如拟牛顿算法（变尺度法）、L-M 算法等，其中拟牛顿算法（变尺度法）的出现有效地弥补了牛顿算法的各种缺陷。

6.4 拟牛顿算法（变尺度法）

在对牛顿算法的改进中，拟牛顿算法诞生了，直到现在该算法还被公认为是求解无约束优化问题的最有效的算法。

6.4.1 算法原理

拟牛顿算法对牛顿算法有两个重要的改进：一是选用对称正定矩阵可以对搜索方向保证下降性质；二是改进变尺度矩阵，通过逐步迭代修正产生，从而避开逐点计算二阶偏导数的大量计算。

Hesse 阵的修正方法有多种，这里主要介绍具有代表性的两种方法：DFP 法与 BFGS 法。

设 $\boldsymbol{s}^{(k)}=\boldsymbol{x}^{(k+1)}-\boldsymbol{x}^{(k)}$，$\boldsymbol{y}^{(k)}=\boldsymbol{g}^{(k+1)}-\boldsymbol{g}^{(k)}$，其中 $\boldsymbol{g}(k)$ 为目标函数 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}^{(k)}$ 点处的梯度量，则 DFP 修正公式为：

$$\boldsymbol{H}^{(k+1)}=\boldsymbol{H}^{(k)}-\frac{\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}+\frac{\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\mathrm{T}}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{s}^{(k)}}$$

BFGS 修正公式可由 DFP 修正公式加上下列修正项的乘积 $\boldsymbol{w}^{(k)}\boldsymbol{w}^{(k)\mathrm{T}}$：

$$\boldsymbol{w}^{(k)}=(\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)})^{1/2}\left(\frac{\boldsymbol{s}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{s}^{(k)}}-\frac{\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}\right)$$

6.4.2 算法步骤

拟牛顿算法的具体步骤如下：

步骤 1：给定初始点 $\boldsymbol{x}^{(0)}$、初始矩阵 $\boldsymbol{H}^{(0)}$（通常取单位阵），计算 $\boldsymbol{g}^{(0)}$，令 $k=0$。

步骤 2：令 $\boldsymbol{p}^{(k)}=-\boldsymbol{H}^{(k)}\boldsymbol{g}^{(k)}$。

步骤 3：由一维搜索（精确或不精确）确定步长 α_k。

$$f(\boldsymbol{x}^{(k)}+\alpha_k\boldsymbol{p}^{(k)})=\min_{\alpha\geq 0} f(\boldsymbol{x}^{(k)}+\alpha\boldsymbol{p}^{(k)})$$

步骤 4：令 $\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+\alpha_k\boldsymbol{p}^{(k)}$。

步骤 5：若 $\|\boldsymbol{g}^{(k+1)}\|\leqslant\varepsilon$，则 $\boldsymbol{x}^*=\boldsymbol{x}^{(k+1)}$ 停；否则令

$$\boldsymbol{s}^{(k)}=\boldsymbol{x}^{(k+1)}-\boldsymbol{x}^{(k)},\ \boldsymbol{y}^{(k)}=\boldsymbol{g}^{(k+1)}-\boldsymbol{g}^{(k)}$$

步骤 6：由 DFP 修正公式

$$\boldsymbol{H}^{(k+1)}=\boldsymbol{H}^{(k)}-\frac{\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}+\frac{\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\mathrm{T}}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{s}^{(k)}}$$

或由 BFGS 修正公式

$$\boldsymbol{w}^{(k)}=(\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)})^{1/2}\left(\frac{\boldsymbol{s}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{s}^{(k)}}-\frac{\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}\right)$$

$$\boldsymbol{H}^{(k+1)}=\boldsymbol{H}^{(k)}-\frac{\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{H}^{(k)}\boldsymbol{y}^{(k)}}+\frac{\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)\mathrm{T}}}{\boldsymbol{y}^{(k)\mathrm{T}}\boldsymbol{s}^{(k)}}+\boldsymbol{w}^{(k)}\boldsymbol{w}^{(k)\mathrm{T}}$$

得到 $\boldsymbol{H}^{(k+1)}$。令 $k=k+1$ 转步骤 2。

6.4.3　算法性质

DFP 与 BFGS 算法的重要性质有：

（1）对正定二次函数，精确一维搜索有二次终结性，且对任意初始对称矩阵 $\boldsymbol{H}^{(0)}$ 有 $\boldsymbol{H}^{(n+1)}=\boldsymbol{G}^{-1}$。

（2）在（1）的前提下，迭代变量满足拟牛顿方程。

（3）在（1）的前提下算法产生共轭的方向，当 $\boldsymbol{H}^{(0)}=\boldsymbol{I}$ 时，产生的梯度相互共轭。

对一般函数还具有以下性质：

（1）在一般条件下，算法保持 $\boldsymbol{H}^{(k)}$ 的对称正定，故有下降性质。

（2）迭代步的计算量（乘法）为 $3n^2+o(n)$。

（3）在一定条件下超线性收敛。

（4）对于严格凸函数用一维搜索，具有全局收敛性。

除上述 DFP 法与 BFGS 法的共同特点外，BFGS 法结合 Wolfe-Powell 不精确一维搜索，在理论上得到全局收敛性，DFP 法至今没有证明有该性质，数值实验进一步验证了该观点。总体上来说 BFGS 法优于 DFP 法，是至今“最好”的拟牛顿算法。

6.4.4　程序示例

MATLAB 实现拟牛顿算法的函数为 fminunc。fminunc 函数是 MATLAB 求解

无约束优化问题的主要函数，拟牛顿算法是 fminunc 函数所使用的最主要方法。有关 Fminunc 的语法说明可以参看 6.2 中的具体说明。

MATLAB 函数的使用方法为：

（1）目标函数程序 BanaFun. m 与 BanaFunWithGrad. m

```
function f = BanaFun(x)（不含导数解析式）
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
function [f,g] = BanaFunWithGrad(x)（含导数解析式）
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
g = [100 * (4 * x(1)^3 - 4 * x(1) * x(2)) + 2 * x(1) - 2; 100 * (2 * x(2) - 2 * x(1)^2)];
```

（2）参数设置 quasi _ newton. m

DFP(Davidon-Fletcher-Powell)方法：

```
OPTIONS = optimset('LargeScale','off','HessUpdate','dfp','gradobj','on',
'MaxFunEvals',250,'InitialHessType','identity','display','iter');
```

BFGS(Broyden-Fletcher-Goldfarb-Shanno)方法：

```
OPTIONS = optimset('LargeScale','off','HessUpdate','bfgs','gradobj','on',
'MaxFunEvals',250,'InitialHessType','identity','display','iter');
```

具体参看附录 Optimization option 参数设置。

'LargeScale','off'	大规模计算模式　关闭
'HessUpdate','dfp'	Hesse 阵修正方式　DFP 方法修正
'HessUpdate','bfgs'	Hesse 阵修正方式　bfgs 方法修正
'gradobj','on'	目标函数导数解析式 on 使用，off 不使用(差分导数)
'MaxFunEvals',250	最大目标函数计算次数　250 次
'InitialHessType','identity'	初始 Hesse 阵使用单位阵(对角元素为 1，其他为 0 的矩阵)
'display','iter'	显示迭代过程

（3）函数计算

1）DPF 方法

```
OPTIONS = optimset('LargeScale','off','HessUpdate','dfp','gradobj','on',
'MaxFunEvals',250,'InitialHessType','identity','display','iter');
x = [-1.9,2]; 初始迭代点
[x,fval,exitflag,output] = fminunc(@BanaFunWithGrad,x,OPTIONS)
```

计算结果为：

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	1	267.62		1.23e+003
1	2	214.416	0.000813405	519
2	7	1.42724	0.0227842	7.46
3	9	1.24439	0.168253	2.48
4	12	1.06017	91	2.14
5	14	0.957704	0.31929	7.37
6	15	0.948335	1	4.94
7	16	0.941972	1	5.49
⋮	⋮	⋮	⋮	⋮
41	60	0.000712298	1	0.399
42	61	0.000118493	1	0.131
43	62	6.82877e-006	1	0.0719
44	63	8.52203e-008	1	0.00219
45	64	1.27563e-009	1	0.00115

Optimization terminated: relative infinity-norm of gradient less than options. TolFun.

x =　　1.0000　　1.0000

fval = 1.2756e-009

exitflag = 　1　(正常停止达到最优解)

output =

　　iterations: 45

　　funcCount: 64

　　stepsize: 1

　　firstorderopt: 0.0011

　　algorithm: 'medium-scale: Quasi-Newton line search'

　　message: [1x85 char]

2) BFGS 方法

```
OPTIONS = optimset('LargeScale','off','HessUpdate','bfgs','gradobj',
'on','InitialHessType','identity','MaxFunEvals',250,'display','iter');
x = [-1.9,2];
[x,fval,exitflag,output] = fminunc(@BanaFunWithGrad,x,OPTIONS)
```

计算结果为:

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	1	267.62		1.23e+003
1	2	214.416	0.000813405	519
2	7	1.42881	0.0226339	7.5
3	9	1.24422	0.168155	2.48
4	12	1.05985	91	2.14
5	14	0.924515	0.352138	7.59
6	15	0.769915	1	2.89
7	16	0.587502	1	6.08
8	17	0.449275	1	1.59
⋮	⋮	⋮	⋮	⋮
22	34	4.97449e-006	1	0.0607
23	35	1.1747e-008	1	0.000357

```
Optimization terminated: relative infinity-norm of gradient less than options. TolFun.
x =      0.9999      0.9998
fval =   1.1747e-008
exitflag =       1
output =
        iterations: 23
        funcCount: 35
        stepsize: 1
        firstorderopt: 3.5671e-004
        algorithm:'medium-scale: Quasi-Newton line search'
        message: [1x85 char]
```

两种算法比较：BFGS 法其迭代次数、目标函数的计算次数都小于 DFP 法，相比较而言 BFGS 法在 bana 函数上效果优于 DFP 法。

6.5 单纯形法

直接算法可以归为经验方法，一般来说不如前面讨论的算法有效。尽管如此，由于它们基本原则的简明性，某些技术的实用性，对于处理某类问题仍有实际意义，对于算法的发展也有着重要作用。

单纯形法（Nelder-Mead Simplex）是直接算法中最常用的一种。单纯形法是利用变换单纯形来求到极小点的方法，是由 Spendley，Hext，HimsWorth（1962）提出的，1964 年 Nelder，Mead 提出了改进的可变多面体算法。

6.5.1　算法原理

单纯形法的基本思想是：

设 $\boldsymbol{x}^{(0)}$，$\boldsymbol{x}^{(1)}$，…，$\boldsymbol{x}^{(n)}$ 是 $\boldsymbol{R}^n$ 中的 $n+1$ 个点，构成一个当前的单纯形，$\boldsymbol{x}_{\max}$，$\boldsymbol{x}_{\min}$定义如下

$$f(\boldsymbol{x}_{\max}) = \max\{f(\boldsymbol{x}^{(0)}), f(\boldsymbol{x}^{(1)}), \cdots, f(\boldsymbol{x}^{(n)})\}$$
$$f(\boldsymbol{x}_{\min}) = \min\{f(\boldsymbol{x}^{(0)}), f(\boldsymbol{x}^{(1)}), \cdots, f(\boldsymbol{x}^{(n)})\}$$

记 $\bar{\boldsymbol{x}}$ 为这个单纯形除去 $\boldsymbol{x}_{\max}$外的所有顶点的形心，

$$\bar{\boldsymbol{x}} = \frac{1}{n}\left(\sum_{i=0}^{n}\boldsymbol{x}^{(i)} - \boldsymbol{x}_{\max}\right)$$

取 $\boldsymbol{x}_{\max}$关于 $\bar{\boldsymbol{x}}$ 的反射点 $\boldsymbol{x}^{(n+1)}$，$\boldsymbol{x}^{(n+1)} = \bar{\boldsymbol{x}} + (\bar{\boldsymbol{x}} - \boldsymbol{x}_{\max})$ 构成新的单纯形，反复上述过程，直到达到停止条件。

算法的几何描述如图 6-3 所示。

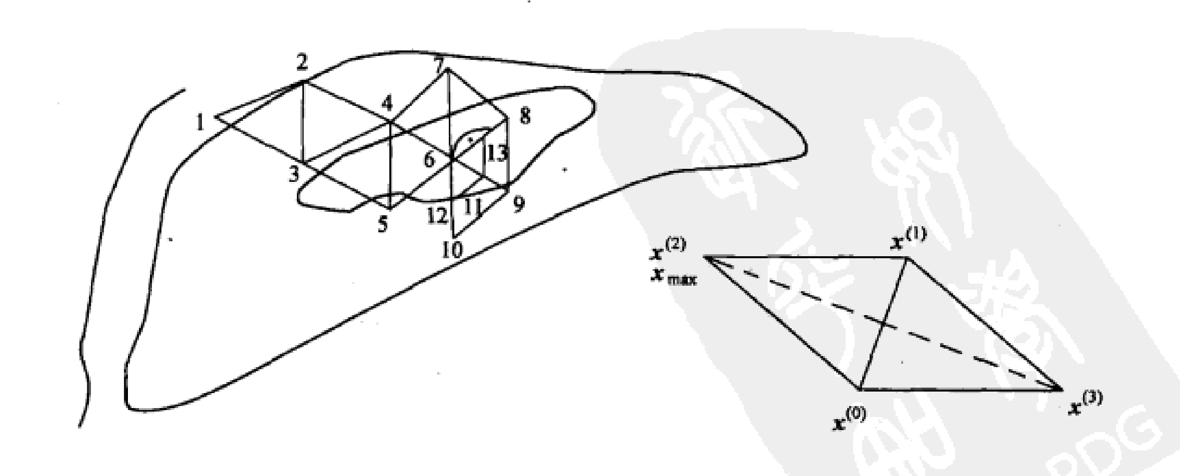


图　6-3

可变多面体算法的具体步骤这里不再详细说明。

6.5.2 函数 Fminsearch

fminsearch 是 MATLAB 中求解无约束的函数之一，其使用的算法为可变多面体算法。

1. 函数语法

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options)
[x,fval] = fminsearch(...)
[x,fval,exitflag] = fminsearch(...)
[x,fval,exitflag,output] = fminsearch(...)
```

函数输入：

Fun：目标函数

X0：迭代初始点

Options：函数参数设置

2. 函数输出

X：最优点(算法停止点)

Fval：最优点对应的函数值

Exitflag：函数停止信息

1：函数收敛正常停止

0：迭代次数，目标函数计算次数达到最大数

-1：算法被输出函数停止(output)

Output：函数运算信息

3. 函数使用

(1) 目标函数程序 BanaFun.m

```
function f = BanaFun(x)
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
```

Nelder-Mead Simplex 函数不需要导数信息。

(2) 算法参数设置：simplexUnc.m

```
OPTIONS = optimset('LargeScale','off','gradobj','off','MaxFunEvals',250,'display','iter');
```

(3) 函数调用运算：simplexUnc.m

```
OPTIONS = optimset('LargeScale','off','gradobj','on','MaxFunEvals',250,'display','iter');
x = [-1.9,2];
[x,fval,exitflag,output] = fminsearch(@BanaFun,x,OPTIONS)
```

4. 计算结果

teration	Func-count	min f(x)	Procedure
0	1	267.62	
1	3	236.42	initial simplex
2	5	67.2672	expand
3	7	12.2776	expand
4	8	12.2776	reflect
5	10	12.2776	contract inside
6	12	6.76772	contract inside
7	13	6.76772	reflect
⋮	⋮	⋮	⋮
110	203	1.86825e-009	contract inside
111	205	1.86825e-009	contract outside
112	207	5.53435e-010	contract inside
113	208	5.53435e-010	reflect
114	210	4.06855e-010	contract inside

```
Optimization terminated:
the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004
and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-004
x =       1.0000      1.0000
fval =    4.0686e-010
exitflag =       1
output =
          iterations: 114
          funcCount: 210
          algorithm:'Nelder-Mead simplex direct search'
          message: [1x196 char]
```

6.6　含参数的优化问题

在实际问题中，优化常常是在一定参数给定的情况下对变量函数进行优化，有时这种含参数的优化问题还会在其他算法的迭代过程中出现，例如一般约束算法中都会有以无约束算法为单元的迭代过程。例如：

$$\min_{x} f(x,a) = a_1 x_1^2 + a_2 x_2^2$$

只有在参数 a 给定的条件下才能计算函数 $f(x,a)$ 的最小点。

Matlab 函数的使用方法为：

(1) 目标函数程序

```
function f = ObjFunWithPara(x,a)
f = a(1) * sin(x(1)) + a(2) * x(2)^2;
```

(2) 函数调用。在函数示例中，不再设置算法参数，各个算法函数均使用默认，可以参看附录 Optimization option, Fminunc 的默认方法为 BFGS。

1) 函数 Fminunc

```
a = [1,1];      % 参数给定
x0 = [0,0];     % 迭代初始点
[x,fval,exitflag,output] = fminunc(@(x) ObjFunWithPara(x,a),x0)
```

计算结果为：

```
Optimization terminated: relative infinity-norm of gradient less than options. TolFun.
x =      -1.5708     0.0000
fval =   -1.0000
exitflag =     1
output =
        iterations: 4
        funcCount: 18
        stepsize: 1
        firstorderopt: 1.9744e-006
        algorithm: 'medium-scale: Quasi-Newton line search'
        message: [1x85 char]
```

2) 函数 Fminsearch

```
a = [1,1];
x0 = [0,0];
[x,fval,exitflag,output] = fminsearch(@(x) ObjFunWithPara(x,a),x0)
```

计算结果为：

```
x =      -1.5708     0.0000
fval =   -1.0000
exitflag =     1
output =
        iterations: 61
```

```
funcCount: 113
algorithm:'Nelder-Mead simplex direct search'
message: [1x196 char]
```

6.7　大规模无约束优化问题

在计算大规模的无约束问题时，都会对原有算法采用一些有效的数值处理技术。由于该类数值计算比较复杂，这里不再详述，以下仅介绍具体的使用方法。

1. 测试函数

$$\min f = \sum_{i=1}^{200}\left(x(i) - \frac{1}{i}\right)^2$$

大规模算法使用的方法主要是 OPTIONS = optimset('LargeScale','on')。

'LargeScale','on'：为算法函数开启大规模计算功能。

2. Fminunc 函数的使用

（1）目标函数程序：LargObjFun. m

```
function f = LargObjFun(x)
f=0;
for i =1:200
  f=f+(x(i) -1/i)^2;
end
```

（2）函数调用 largUnc. m

```
x0 =10 * ones(1,200)（初始迭代点）
PTIONS = optimset('LargeScale','on','display','iter','TolFun',1e-8);
[x,fval,exitflag,output] = fminunc(@LargObjFun,x0,PTIONS)
```

计算结果为：

Iteration	Func-count	f(x)	Step-size	optimality
0	201	19884.1		20
1	402	16104.3	0.050025	18
2	603	6.31478e-007	1	0.000287
3	804	1.11396e-014	1	8.9e-010

```
Optimization terminated: relative infinity-norm of gradient less than options. TolFun.
x =
  Columns 1 through 9
```

```
    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429
    0.1250    0.1111
      ⋮         ⋮         ⋮         ⋮         ⋮         ⋮         ⋮
  Columns 190 through 198
    0.0053    0.0052    0.0052    0.0052    0.0052    0.0051    0.0051
    0.0051    0.0051
  Columns 199 through 200
    0.0050    0.0050
fval = 1.1140e-014
exitflag = 1
output =
        iterations: 3
         funcCount: 804
          stepsize: 1
     firstorderopt: 8.8988e-010
         algorithm: 'medium-scale: Quasi-Newton line search'
           message: [1x85 char]
```

第 7 章　约束优化算法

约束优化问题是人们在日常生活中遇到最多的数学问题。一般的约束优化问题的求解难度很大,目前还没有一种普遍有效的方法。本章介绍一般约束问题的经典算法。由于算法的复杂性,文中只介绍基本解法,然后对 MATLAB 求解约束优化的函数进行详细的介绍。

约束优化问题的标准形式为：

$$\min f(\boldsymbol{x}),\ \boldsymbol{x}\in\mathbf{R}^n$$
$$\text{s.t.}\quad g_i(\boldsymbol{x})\leqslant 0,\ i=1,2,\cdots,m$$
$$h_j(\boldsymbol{x})=0,\ j=1,2,\cdots,l$$

其中$f,\ g_i,\ h_j:\mathbf{R}^n\rightarrow\mathbf{R}$

约束优化算法的基本思想是：通过引入效用函数的方法将约束优化问题转换成无约束问题，再利用优化迭代过程不断地更新效用函数，以使得算法收敛。本章中的无约束算法主要介绍的是内点罚函数法和拉格朗日乘子法，其中拉格朗日乘子法是在一般的罚函数法的基础上建立起来的。它引入了拉格朗日乘子，成功地解决了原罚函数法增广目标函数的病态性，即原罚因子趋向无穷大（或零）的病态性质。

7.1　罚函数法（内点法）

罚函数法（内点法）的主要思想是：在可行域的边界上筑起一道很高的“围墙”，当迭代点靠近边界时，目标函数陡然增大，以示惩罚，阻止迭代点穿越边界，这样就可以将最优解“挡”在可行域之内了。

它只适用于不等式约束：

$$\min f(\boldsymbol{x}),\ \boldsymbol{x}\in\mathbf{R}^n$$
$$\text{s.t.}\quad g_i(\boldsymbol{x})\leqslant 0,\ i=1,2,\cdots,m$$

它的可行域为：

$$D=\{\boldsymbol{x}\in\mathbf{R}^n \mid g_i(\boldsymbol{x})\leqslant 0,\ i=1,2,\cdots,m\}$$

对上述约束问题，且其可行域的内点可行集$D_0\neq\varnothing$的情况下，引入效用函数：

$$\min B(\boldsymbol{x},r)=f(\boldsymbol{x})+r\bar{B}(\boldsymbol{x})$$

其中 $\widetilde{B}(\boldsymbol{x}) = \sum_{i=1}^{m} -\frac{1}{g_i(\boldsymbol{x})}$ 或 $\widetilde{B}(\boldsymbol{x}) = \sum_{i=1}^{m} |\ln(-g_i(\boldsymbol{x})|$。

算法的具体步骤如下：

给定控制误差 $\varepsilon>0$，惩罚因子的缩小系数 $0<c<1$。

步骤 1：令 $k=1$，选定初始点 $\boldsymbol{x}^{(0)} \in D_0$，给定 $r_1>0$(一般取 10)。

步骤 2：以 $\boldsymbol{x}^{(k)}$ 为初始点，求解无约束

$$\min B(\boldsymbol{x}, r) = f(\boldsymbol{x}) + r_k \widetilde{B}(\boldsymbol{x})$$

其中 $\widetilde{B}(\boldsymbol{x}) = \sum_{i=1}^{m} -\frac{1}{g_i(\boldsymbol{x})}$ 或 $\widetilde{B}(\boldsymbol{x}) = \sum_{i=1}^{m} |\ln(-g_i(\boldsymbol{x}))|$，得最优解 $\boldsymbol{x}^{(k)} = \boldsymbol{x}(r_k)$。

步骤 3：若 $r_k \widetilde{B}(\boldsymbol{x}^{(k)}) < \varepsilon$，则 $\boldsymbol{x}^{(k)}$ 为其近似最优解，停；否则，令 $r_k = cr_k$，$k=k+1$，转步骤 2

7.2 拉格朗日乘子法

这里介绍拉格朗日乘子法处理不等式约束的两种方法及其比较，并引入经典的松弛变量方法。

1. PH 算法：(约数为等式的情况下)引入

效用函数为：

$$\min M(\boldsymbol{x}, \boldsymbol{u}^{(k)}, \sigma_k) = f(\boldsymbol{x}) + \boldsymbol{u}^{(k)\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \sigma_k \boldsymbol{h}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})$$

判断函数为：

$$\phi_k = \|\boldsymbol{h}(\boldsymbol{x}^{(k)})\|$$

当 $\phi_k = \phi(\boldsymbol{x}^{(k)}) < \varepsilon$ 时迭代停止。

步骤 1：选定初始点 $\boldsymbol{x}^{(0)}$，初始拉格朗日乘子向量 $\boldsymbol{u}^{(1)}$，初始罚因子 σ_1 及其放大系数 $c>1$，控制误差 $\varepsilon>0$ 与常数 $\theta \in (0, 1)$，令 $k=1$。

步骤 2：以 $\boldsymbol{x}^{(k+1)}$ 为初始点，求解无约束问题：

$$\min M(\boldsymbol{x}, \boldsymbol{u}^{(k)}, \sigma_k) = f(\boldsymbol{x}) + \boldsymbol{u}^{(k)\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \sigma_k \boldsymbol{h}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})$$

得到无约束问题最优解 $\boldsymbol{x}^{(k)}$。

步骤 3：当 $\|h(\boldsymbol{x}^{(k)})\| < \varepsilon$ 时，$\boldsymbol{x}^{(k)}$ 为所求的最优解，停；否则转步骤 4。

步骤 4：当 $\|h(\boldsymbol{x}^{(k)})\| / \|h(\boldsymbol{x}^{(k)})\| < \theta$ 时，转步骤 5；否则令 $\sigma_{k+1} = c\sigma_k$，转步骤 5。

步骤 5：令 $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \sigma_k \boldsymbol{h}(\boldsymbol{x}^{(k)})$，$k=k+1$，转步骤 1。

2. PHR 算法(一般约束形式的松弛变量法和指数形式法)

松弛变量法为：

$$M(u, v, \rho) = f(x) + \frac{1}{2\rho}\sum_{i=1}^{m}\{[\max(0, u_i + \rho g_i(x))]^2 - u_i^2\}$$
$$+ \sum_{j=1}^{l} v_j h_j(x) + \frac{\rho}{2}\sum_{j=1}^{l} h_j^2(x)$$

乘子的修正公式为：

$$v_j^{(k+1)} = v_j^{(k)} + \rho h_j(x^{(k)}),\ j = 1, \cdots, l$$
$$u_i^{(k+1)} = \max[0, u_i^{(k)} + \rho g_i(x^{(k)})],\ i = 1, \cdots, m$$

判断函数为：

$$\phi_k = \left\{\sum_{j=1}^{l} h_j^2(x^{(k)}) + \sum_{i=1}^{m}\max\left(-g_i(x^{(k)}), \frac{u_i^{(k)}}{\rho}\right)^2\right\}^{\frac{1}{2}}$$

当 $\phi_k = \phi(x^{(k)}) < \varepsilon$ 时迭代停止。

7.3　乘子法 MATLAB 程序及其使用

7.3.1　Al _ main 函数

根据上述7.2乘子法利用m语言编写的乘子法示例程序如下：

1. 程序(1)：乘子法效用函数程序

函数功能：将约束优化问题，根据效用函数方法，将其转变成无约束问题。

```
function f = AL _ obj(x)
% 拉格朗日增广函数
% N _ equ   等式约束个数
% N _ inequ 不等式约束个数
global r _ al pena N _ equ N _ inequ;% 全局变量
h _ equ = 0;
h _ inequ = 0;
[h,g] = constrains(x);
% 等式约束部分;
for i = 1:N _ equ
   h _ equ = h _ equ  + h(i) * r _ al(i) + (pena/2) * h(i).^2;
end
% 不等式约束部分
for i = 1:N _ inequ
```

```
    h_inequ = h_inequ + (0.5/pena) * (max(0,(r_al(i) +
        pena * g(i))).^2 - r_al(i).^2);
end
%拉格朗日增广函数值
f = obj(x) + h_equ + h_inequ;
```

2. 程序(2)：判断函数

函数功能：判断是否符合约束条件。

```
%% the compare function is the stop condition
function f = compare(x)
global r_al pena N_equ N_inequ;
h_equ = 0;
h_inequ = 0;
[h,g] = constrains(x);
%等式部分
for i = 1:N_equ
    h_equ = h_equ + h(i).^2;
end
%不等式部分
for i = 1:N_inequ
    h_inequ = h_inequ + (max( -g(i), r_al(i + N_equ)/pena)).^2;
end
f = sqrt(h_equ  + h_inequ);
```

3. 程序(3)：AL 算法主程序

函数功能：对无约束的效用函数利用拟牛顿算法求解其最优解，更新乘子。

```
function [X,FVAL] = AL_main(x_al,r_al,N_equ,N_inequ)
%code by ariszheng
%Email: ariszheng@gmail.com
%本程序为拉格朗日乘子法示例算法
%函数输入:
%      x_al: 初始迭代点
%      r_al: 初始拉格朗日乘子
%      N_equ:   等式约束个数
%      N_inequ: 不等式约束个数
%函数输出:
```

```
%      X:     最优函数点
%      FVAL: 最优函数值
% ==============程序开始========================
global r_al pena N_equ N_inequ; %参数(全局变量)
pena = 10;%惩罚系数
c_scale = 2;%乘法系数乘数;
cta = 0.5; %下降标准系数
e_al = 0.005;%误差控制范围
max_itera = 25;
out_itera = 1;%迭代次数
% ==================算法迭代开始====================
while out_itera < max_itera
    x_al0 = x_al;
    r_al0 = r_al;
    %判断函数
    compareFlag = compare(x_al0);
    %无约束的拟牛顿法 BFGS
    [X,FVAL] = fminunc(@AL_obj,x_al0);
    x_al = X;%得到新迭代点
    % 判断停止条件
    if compare(x_al) < e_al
      disp('we get the opt point:');
      break
    end
    %c 判断函数下降度
    if compare(x_al) < cta * compareFlag
      pena = pena;%可以根据需要修改惩罚系数变量
    else
      pena = min(1000, c_scale * pena);%% 乘法系数最大 1000;
      disp('pena = 2 * pena');
    end
    %% 更新拉格朗日乘子;
    [h,g] = constrains(x_al);
    for i = 1:N_equ
```

```
        %% 等式约束部分;
        r_al(i) = r_al(i) + pena * h(i);
    end
    for i = 1:N_inequ
        %% 不等式约束部分;
        r_al(i + N_equ) = max(0, (r_al(i + N_equ) + pena * g(i)));
    end
    out_itera = out_itera + 1;
end
% +++++++++++++++++++++迭代结束+++++++++++++++++
disp('!!!!!!!!!!!!!!!!!!the iteration over!!!!!!!!!!!!!!!!!!!!!!!');
disp('the value of the obj function');
obj(x_al)
disp('the value of constrains');
compare(x_al)
disp('the opt point');
    X = x_al;
    FVAL = obj(X);
```

7.3.2 乘子法 Al_main 函数使用方法

1. 定义目标函数及约束函数

$$\min f(x) = -x_1x_2 - x_2x_3 - x_3x_1$$
$$\text{s.t. } x_1 + x_2 + x_3 - 1 = 0$$
$$x_1^2 + x_2^2 + x_3^2 - 3 \leqslant 0$$

目标函数 m 文件：(Augmented Lagrangian Algorithms 文件夹)

```
function f = obj(x)
f = -x(1) * x(2) - x(2) * x(3) - x(3) * x(1);
```

约束函数 M 文件：

```
function [h,g] = constrains(x)
h = x(1) + x(2) + x(3) - 1;% 等式约束
g = x(1)^2 + x(2)^2 + x(3)^2 - 3;% 不等式约束
```

2. AL_main 函数调用

```
function [X,FVAL] = AL_main(x_al,r_al,N_equ,N_inequ)
```

函数输入：

x_al：初始迭代点

r_al：初始拉格朗日乘子

N_equ：　等式约束个数

N_inequ：　不等式约束个数

函数输出：

X：　　最优函数点

FVAL：最优函数值

函数调用：

```
x_al = [1,1,1]; 初始迭代点
r_al = [1,1];    初始拉格朗日乘子
N_equ = 1;       等式约束个数 一个
N_inequ = 1;     不等式约束个数 一个
[X,FVAL] = AL_main(x_al,r_al,N_equ,N_inequ)
```

计算结果：

```
the opt point
X =
    0.3342    0.3342    0.3342
FVAL =
    -0.3350
```

7.4　Fmincon 函数

Fmincon 是 MATLAB 最主要的内置求解约束最优化的函数，改函数的优化问题的标准形式为：

$$
\begin{aligned}
&\min_{x} f(\boldsymbol{x}) \\
&\text{s.t. } \boldsymbol{c}(\boldsymbol{x}) \leqslant 0 \\
&\boldsymbol{ceq}(\boldsymbol{x}) = 0 \\
&\boldsymbol{Ax} \leqslant \boldsymbol{b} \\
&\boldsymbol{Aeqx} = \boldsymbol{beq} \\
&\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}
\end{aligned}
$$

这里 $\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{beq}$, $\boldsymbol{lb}$, $\boldsymbol{ub}$ 为向量；$\boldsymbol{A}$ 与 $\boldsymbol{Aeq}$ 为矩阵；$f(\boldsymbol{x})$ 为目标函数；$\boldsymbol{c}(\boldsymbol{x})$，$\boldsymbol{ceq}(\boldsymbol{x})$ 为非线性约束；$\boldsymbol{Ax} \leqslant \boldsymbol{b}$，$\boldsymbol{Aeqx} = \boldsymbol{beq}$ 为线性约束；$\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$ 为可行解的区间约束。

Fmincon 函数使用的约束优化算法都是目前比较适用的有效算法，对于中等的约束优化问题，fmincon 使用序列二次规划（Sequential Quadratic Programming，SQP）算法，对于大规模约束优化问题，fmincon 使用基于内点反射牛顿法的信赖域算法（subspace trust region method and is based on the interior-reflective Newton method），对于大规模的线性系统，使用共轭梯度算法（Preconditioned Conjugate Gradients，PCG）。由于这些算法都具有一定的复杂性，具体算法这里不再详述。

函数语法

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
[x,fval] = fmincon(…)
[x,fval,exitflag] = fmincon(…)
[x,fval,exitflag,output] = fmincon(…)
[x,fval,exitflag,output,lambda] = fmincon(…)
[x,fval,exitflag,output,lambda,grad] = fmincon(…)
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(…)
```

函数输入

Fun：目标函数

X0：初始迭代点

A：线性不等约束系数矩阵

B：线性不等式约束的常数向量

Aeq：线性等约束系数矩阵

Beq：线性等式约束的常数向量

lb：可行区域下届

Ub：可行区域上界

Nonlcon：非线性约束

Options：优化参数设置

函数输出

X：最优点(或者结束迭代点)

Fval：最优点(或者结束迭代点对应的函数值)

Exitflag：迭代停止标识

Output：算法输出（算法计算信息等）

Ambda：拉格朗日乘子

Grad：一阶导数向量

Hessian：二阶导数矩阵

计算示例：

使用具体计算示例说明 fmincon 的具体使用方法，在示例中还将对 fmincon 的使用细节加以说明。

7.4.1　函数示例（1）

$$\begin{cases}\min f(\boldsymbol{x}) = -x_1x_2x_3 \\ 0 \leqslant x_1 + 2x_2 + 2x_3 \leqslant 72\end{cases} \Rightarrow \begin{cases}\min f(\boldsymbol{x}) = -x_1x_2x_3 \\ -x_1 - 2x_2 - 2x_3 \leqslant 0 \\ x_1 + 2x_2 + 2x_3 \leqslant 72\end{cases}$$

编写目标函数 M 文件 myfun1. m

```
function f = myfun1(x)
f = -x(1) * x(2) * x(3);
```

调用 fmincon 函数 M 文件 SolveMyfun1. m

```
options = optimset('LargeScale','off','display','iter');
%参数设置使用中等规模算法，显示迭代过程
A = [-1,-2,-2; %线性不等式约束系数矩阵
     1,2,2];
b = [0;72];%线性不等式约束常量向量
x0 = [10,10,10];%初始迭代点
[x,fval,exitflag,output,lambda,grad,hessian] =
fmincon(@myfun1,x0,A,b,[],[],[],[],[],options)
```

计算结果为：

Iter	F-count	f(x)	max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	4	-1000	-22				
1	9	-1587. 17	-11	0. 5	642	584	
2	13	-3323. 25	0	1	-1. 9e+003	161	
3	21	-3324. 69	0	0. 0625	146	58. 2	Hessian modified

```
  ⋮      ⋮        ⋮        ⋮      ⋮          ⋮          ⋮
 10     49     -3456       0      1   -0.000103    0.0487
 11     53     -3456       0      1   -4.39e-007   0.00247
Optimization terminated: magnitude of directional derivative in search
 direction less than 2 * options.TolFun and maximum constraint violation
  is less than options.TolCon.（说明正常终止）
Active inequalities (to within options.TolCon = 1e-006):
  lower      upper     ineqlin    ineqnonlin
                          2
x =24.0000      12.0000      12.0000
fval = -3.4560e+003
exitflag =5
```

（下降方向的范数小于误差控制范围，正常迭代停止，Magnitude of directional derivative was less than the specified tolerance and constraint violation was less than options. TolCon. ）

```
output =
     iterations: 11
     funcCount: 53
     lssteplength: 1
     stepsize: 0.0011
     (算法 中等规模 SQP 拟牛顿 线性搜索)
     algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
     firstorderopt: 4.7583e-004
     message: [1x172 char]
lambda =
         lower: [3x1 double]
         upper: [3x1 double]
         eqlin: [0x1 double]
      eqnonlin: [0x1 double]
        ineqlin: [2x1 double]
    ineqnonlin: [0x1 double]
grad =
-144.0002
```

```
-287.9994
-288.0002
hessian =
      3.2718     -3.7384     -5.0124
     -3.7384     22.6255     -5.8906
     -5.0124     -5.8906     15.0341
```

7.4.2　函数示例（2）

fmincon 函数默认的求导方法为差分求导方法。为了便于更精确地计算以及减少计算时间，可以提供目标函数导数。以下还是以函数示例（1）的优化问题为例。

编写目标函数 M 文件 myfun2.m

```
function [f,g] = myfun2(x)
f = -x(1) * x(2) * x(3);
g = [ -x(2) * x(3);
     -x(1) * x(3);
     -x(1) * x(2)];
```

调用 fmincon 函数 M 文件 SolveMyfun2.m

```
options = optimset('LargeScale','off','display','iter');
%参数设置使用中等规模算法，显示迭代过程
A = [ -1, -2, -2; %线性不等式约束系数矩阵
        1, 2, 2];
b = [0;72];%线性不等式约束常量向量
x0 = [10,10,10];%初始迭代点
[x,fval,exitflag,output,lambda,grad,hessian] =
fmincon(@myfun1,x0,A,b,[],[],[],[],[],options)
```

计算结果为：

```
Optimization terminated: magnitude of directional derivative in search
 direction less than 2 * options.TolFun and maximum constraint violation
  is less than options.TolCon.（正常终止）
Active inequalities (to within options.TolCon = 1e-006):
  lower      upper      ineqlin      ineqnonlin
                           2
x =24.0000      12.0000      12.0000
```

```
fval = -3.4560e+003
exitflag = 5
output =
      iterations: 11
      funcCount: 28
      lssteplength: 1
      stepsize: 0.0011
      algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
      firstorderopt: 4.7597e-004
      message: [1x172 char]
```

比较函数示例（1）与函数示例（2），在提供一阶导数后，目标函数计算次数由 53 次降为 28 次，可以有效地减少函数计算时间

7.4.3 函数示例（3）

fmincon 函数默认的求导方法为差分求导方法。为了便于更精确地计算以及减少计算时间，可以提供目标函数导数，fmincon 函数的默认更新 hesse 阵的方法为 BFGS。对于大规模算法，为了便于更精确地计算以及减少计算时间，可以提供目标函数 hesse 阵函数。以下还是以函数示例（1）的优化问题为例。

编写目标函数 M 文件 myfun3.m

```
function [f,g,h] = myfun3(x)
f = -x(1)*x(2)*x(3);
g = [-x(2)*x(3);          %一阶导数
     -x(1)*x(3);
     -x(1)*x(2)];
h = [0, -x(3), -x(2);     %二阶导数
     -x(3), 0, -x(1);
     -x(2), -x(1), 0];
```

调用 fmincon 函数 M 文件 SolveMyfun3.m

```
options = optimset('LargeScale','on','GradObj','on','Hessian','on');
%参数设置使用中等规模算法,使用目标函数一阶与二阶导函数
A = [-1, -2, -2; %线性不等式约束系数矩阵
      1, 2, 2];
b = [0;72];%线性不等式约束常量向量
x0 = [10,10,10];%初始迭代点
```

```
[x,fval,exitflag,output] = fmincon(@myfun3,x0,A,b,[],[],[],[],[],options)
```

计算结果为：

大规模算法-信赖域算法不能解决该类型问题，用中等规模算法代替。显示如下：

```
> In fmincon at 303
  In SolveMyfun3 at 7
(Quasi-Newton method 不需要分析 Hessian 阵)
Warning: Medium-scale method is a Quasi-Newton method and does not use
analytic Hessian. Hessian flag in options will be ignored.
> In fmincon at 308
  In SolveMyfun3 at 7
Optimization terminated: magnitude of directional derivative in search
 direction less than 2 * options.TolFun and maximum constraint violation
  is less than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):
  lower      upper     ineqlin     ineqnonlin
                          2
x =
      24.0000      12.0000      12.0000
fval =
-3.4560e+003
exitflag =
     5
output =
        iterations: 11
         funcCount: 28
      lssteplength: 1
          stepsize: 0.0011
         algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
     firstorderopt: 4.7597e-004
           message: [1x172 char]
```

该示例只做 Hessian 阵使用方法的示例，只有在求解大规模优化问题的时候，Hessian 阵分析方法才被使用。

7.4.4 函数示例（4）

$$\min f = \prod_{i=1}^{n}\left(x(i) - \frac{1}{i}\right),\ n = 100$$

$$\text{s. t.} \qquad -10 \leqslant x(i) \leqslant 10,\ i = 1,\ 2,\ \cdots,\ 100$$

编写目标函数 M 文件 myfun4. m

```
function f = myfun4(x)
n = length(x);% n = 100
f = 1;
for i = 1:100
    f = f * (x(i) - 1/i);
end
```

调用 fmincon 函数 m 文件 SolveMyfun4. m

Iter	F-count	f(x)	max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	101	0	-9				
1	202	-0. 0001	-9	1	-0. 0001	0. 01	
2	303	-0. 013943	-8. 99	1	-0. 0216	0. 0273	
3	404	-0. 263441	-8. 838	1	-0. 671	0. 483	
4	505	-3. 30667e +013	0	1	-8. 99e +014	2. 79e +013	
5	606	-7. 2132e +099	0	1	-6. 18e +101	7. 59e +098	
6	706	-7. 2132e +099	0	2	0	0\	

最优化终止：一阶导的误差小于 TolFun 的限制，并且约束的最大违背要求的数值小于 TolCon 的限制。

```
X = [ -10,10,10,…. 10]
fval = -7. 2132e +099
exitflag = 1
output =
    iterations: 6
    funcCount: 706
```

```
lssteplength: 2
stepsize: 0
algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
firstorderopt: 0
message: [1x144 char]
```

7.4.5　函数示例 (5)

$$\min f = -\sum_{i=1}^{n} \exp(x(i)) \times \sin(x(i)),\ n = 10$$

$$\text{s.t.}\quad \sum_{i=1}^{n} x(i)^2 \leqslant 100$$

$$0 \leqslant x(i) \leqslant 10 \quad i = 1, 2, \cdots, 10$$

编写目标函数 M 文件 myfun5. m

```
function f = myfun5(x)
n = length(x);% n = 100
f = -sum(exp(x). * sin(x));
% -sum(exp(x). * sin(x))
% = -exp(x(1)) * sin(x(1)),…,-exp(x(n)) * sin(x(n))
```

编写非线性约束函数 M 文件 mycon5. m

```
function[c,ceq] = mycon5(x)
c = sum(x.^2) -100; % 关于 x 非线性不等式约束.
% sum(x.^2) -100 = x(1)^2 + … + x(n)^2 -100 <=0
ceq = []; % 关于 x 非线性等式约束.
```

调用 fmincon 函数的 M 文件 solveMyfun5. m

```
options = optimset('LargeScale','off','display','iter');
% 参数设置使用中等规模算法,显示迭代过程
lb = zeros(1,10);% lb = [0,0,....,0]
ub = 10 * ones(1,10); % ub = [10,10,....,10]
x0 = ones(1,10);% x0 = [1,1,....,1]
[x,fval,exitflag,output] = fmincon(@ myfun5,x0,[],[],[],[],lb,ub,@ mycon5,options)
```

计算结果为：

Iter	F-count	f(x)	max constraint	Line search steplength	Directional derivative	First-order optimality	Procedure
0	11	-22.8736	-1				
1	23	-46.3194	-2.878	0.5	471	12.5	
2	35	-71.9827	-2.156	0.5	35.2	2.43	
3	46	-74.1165	-2.273	1	-1.34	1.14	
4	57	-74.5731	-2.377	1	0.324	0.312	
5	68	-74.6046	-2.354	1	0.00591	0.0282	
6	80	-74.6047	-2.353	0.5	0.000148	0.0401	
7	93	-74.6049	-2.357	0.25	0.000639	0.0122	
8	104	-74.6049	-2.356	1	2.14e-007		

最优化终止：搜索方向的方向导数小于 TolFun 限制值的 2 倍，并且最大的违背约束值小于 TolCon 限制值，无起作用的不等式。

x =

Columns 1 through 9：2.3562　2.3562　2.3562　2.3562　2.3562　2.3562　2.3562　2.3562　2.3562；Column 10：2.3562。

fval = -74.6049

exitflag = 5

output =

iterations：8

funcCount：104

lssteplength：1

stepsize：0.0020

algorithm：'medium-scale：SQP，Quasi-Newton，line-search'

firstorderopt：3.4526e-004

message：[1x172 char]

7.4.6 函数示例（6）

函数示例（5）只有不等式约束，现在在示例（5）的约束优化问题中加入等式约束，原优化问题变为：

$$\min f = -\sum_{i=1}^{n} \exp(x(i)) \times \sin(x(i)),\ n = 10$$

$$\text{s.t.}\quad \sum_{i=1}^{n} x(i)^2 \leqslant 100$$

$$\sum_{i=1}^{n} x(i) = 20$$

$$0 \leqslant x(i) \leqslant 10,\ i = 1, 2, \cdots, 10$$

编写目标函数 M 文件 myfun6. m

```
function f = myfun6(x)
n = length(x);% n = 100
f = -sum(exp(x).*sin(x));
% -sum(exp(x).*sin(x))
% = -exp(x(1))*sin(x(1)),…, -exp(x(n))*sin(x(n))
```

编写非线性约束函数 m 文件 mycon6. m

```
function [c,ceq] = mycon6(x)
c = sum(x.^2) - 100; % 关于 x 非线性不等式约束.
% sum(x.^2) - 100 = x(1)^2 + … + x(n)^2 - 100 <= 0
ceq = sum(x) - 20; % 关于 x 非线性等式约束
```

调用 fmincon 函数的 m 文件 solveMyfun6. m

```
options = optimset('LargeScale','off','display','off');
% 参数设置使用中等规模算法，不显示迭代过程
lb = zeros(1,10);% lb = [0,0,…,0]
ub = 10 * ones(1,10); % ub = [10,10,…,10]
x0 = ones(1,10);% x0 = [1,1,…,1]
[x,fval,exitflag,output] = fmincon(@myfun6,x0,[],[],[],[],lb,ub,@mycon6,options)
```

计算结果为：

```
x =
  Columns 1 through 9
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000
2.0000    2.0000
  Column 10
    2.0000
fval = -67.1885
```

```
exitflag = 1
output =
    iterations: 2
    funcCount: 35
    lssteplength: 0.2500
    stepsize: 4.7595e-007
    algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
    firstorderopt: 8.4232e-007
    message: [1x144 char]
```

7.4.7　函数示例（7）

含参数的优化问题：一个控制系统要根据环境作优化配置。如果环境变化，原优化方程中的系数就会发生变化，需要根据新的环境参数重作做优化配置。这个控制系统会用到含参数的优化问题。

$$\min_x f = \sum_{i=1}^{n} (x(i) - a(i))^2,\ n = 5$$

$$\text{s.t.}\ \sum_{i=1}^{n} x(i) = b$$

$$0 \leqslant x(i) \leqslant 10,\ i = 1, 2, \cdots, 5$$

其中 a，b 为参数。

编写目标函数 M 文件 myfun7.m

```
function f = myfun7(x,a)
f = sum((x-a).^2);
%f = (x(1) -a(1))^2 + … + (x(n) -a(n))^2
```

编写约束函数 m 文件 mycon7.m

```
function [c,ceq] = mycon7(x,b)
c = [];
ceq = sum(x) -b; % 关于 x 非线性等式约束.
```

编写调用 fmincon 的 M 文件 solveMyfun7

```
options = optimset('LargeScale','off','display','off');
% 参数设置使用中等规模算法,不显示迭代过程
lb = zeros(1,5);% lb = [0,0,…,0]
ub = 10 * ones(1,5); % ub = [10,10,…,10]
a = [1,2,3,4,5];% 参数 a = [1,2,3,4,5]
```

```
b = 15;     % 参数 b = 15
x0 = ones(1,5);% x0 = [1,1,…,1]
[x,fval,exitflag,output] = fmincon(@(x) myfun7(x,a), x0, [], [], [], [], lb,
ub, @(x)mycon7(x,b), options)
```

计算结果为：

```
x =
    1.0000    2.0000    3.0000    4.0000    5.0000
fval = 7.5765e-014
exitflag = 1
output =
    iterations: 6
    funcCount: 42
    lssteplength: 1
    stepsize: 3.0857e-004
    algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
    firstorderopt: 5.7251e-007
    message: [1x144 char]
```



第 8 章　非线性最小二乘法

在数据拟合或者曲线拟合中都会用到最小二乘法，即残差平方和最小的方法。其标准模型为：

$$\min f(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m}[r_i(\boldsymbol{x})]^2 = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2 \tag{8-1}$$

其中 $\boldsymbol{r}(\boldsymbol{x}) = (r_1(\boldsymbol{x}), \cdots, r_m(\boldsymbol{x}))^{\mathrm{T}}$ 称为在 x 点的残差向量。设拟合数据样本集合为：$X = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}\}$，$Y = \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \cdots, \boldsymbol{y}^{(m)}\}$；回归模型为：$\boldsymbol{y} = \boldsymbol{L}(\boldsymbol{x})$，则残差项为 $\boldsymbol{r}(\boldsymbol{x}^{(i)}) = \boldsymbol{L}(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)}$。

数据拟合根据拟合模型的性质以及约束条件，可以分为线性拟合与非线性拟合。由于无约束线性拟合的解法已经非常成熟，所以本文主要讨论带约束的以及非线性模型的最优系数求解方法。非线性最小二乘法的求解基本思路是：把非线性最小二乘法问题当作非线性最优化问题，再结合非线性最小二乘法的特殊数学性质，引入新的专门针对非线性最小二乘法的非线性最优化方法，即高斯-牛顿法(Guass-Newton)，莱温伯格-马奎特法(Levenberg-Marquardt)，简称 L-M 法。

MATLAB 的内置求解非线性最小二乘法函数包括：

（1）lsqnonneg：求解非负约束的最小二乘问题。

（2）lsqlin：求解带约束的线性最小二乘问题。

（3）lsqnonlin：求解非线性最小二乘问题。

（4）lsqcurvefit：求解非线性数据拟合问题，在最小二乘原理情况下使用。

8.1　高斯-牛顿法

对于非线性最小二乘问题：

$$\min f(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m}[r_i(\boldsymbol{x})]^2$$

根据上两章关于非线性最优化的介绍，我们知道目标函数的一阶函数与二阶导数矩阵对优化的整个迭代过程起着至关重要的作用。如果函数 $\boldsymbol{r}(\boldsymbol{x}) = (r_1(\boldsymbol{x}),\cdots,r_m(\boldsymbol{x}))^{\mathrm{T}}$ 是二次连续可微，则 $f(\boldsymbol{x})$ 的一阶导数与二阶导数(hessian 阵)分别为：

$$\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x})$$

$$G(x)=\nabla^2 f(x)=A(x)A(x)^{\mathrm{T}}+\sum_{i=1}^{m} r_i(x)\,\nabla^2 r_i(x)$$
$$=M(x)+S(x)$$

这里　　　　　　$A(x)=[\nabla r_1(x),\cdots,\nabla r_m(x)]$

对一般无约束优化问题的高斯-牛顿算法，在迭代点 $x^{(k)}$ 取目标函数 $f(x)$ 下列形式的二次近似：

$$q(\delta^{(k)})=f(x^{(k)})+g^{(k)\mathrm{T}}\delta^{(k)}+\frac{1}{2}\delta^{(k)\mathrm{T}}M^{(k)}\delta^{(k)}$$

的最优化解作为搜索方向，利用 $M(x)$ 近似代替 $B(x)$。

$$q(\delta^{(k)})-f(x^{(k)})=(A(\delta^{(k)})r(\delta^{(k)}))^{\mathrm{T}}\delta^{(k)}+\frac{1}{2}\delta^{(k)\mathrm{T}}(A(\delta^{(k)})A(\delta^{(k)})^{\mathrm{T}})\delta^{(k)}$$

的最优化解为：

$$A(\delta^{(k)})A(\delta^{(k)})^{\mathrm{T}}\delta^{(k)}=-A(\delta^{(k)})r(\delta^{(k)})\text{的解。}$$

下降迭代为：

$$x^{(k+1)}=\alpha_k\delta^{(k)}$$

其中 α_k 为步长，可以通过线性搜索确定。

从上述高斯-牛顿法基本思想可以得出该算法的以下三个特性：

(1) 高斯-牛顿法的收敛与否直接取决于 hessian 阵的缺省部分 $S(x)$ 的大小，如果 $S(x)$ 大于 $M(x)$，则高斯-牛顿法一般不收敛。

(2) 如果 $M(x)$ 正定，则高斯-牛顿法局部收敛。

(3) 如果算法收敛，则收敛速度基本取决于 $S(x)$ 的大小。$S(x)=0$，算法二次收敛，$S(x)\neq 0$ 算法线性收敛。

根据高斯-牛顿法这三个特性，人们引入了许多改善高斯-牛顿法的方法，其中比较有效的为阻尼牛顿法与莱温伯格-马奎特法。

莱温伯格-马奎特法对高斯-牛顿法的改善是通过将高斯-牛顿法的近似求解下降方向公式：

$$A(\delta^{(k)})A(\delta^{(k)})^{\mathrm{T}}\delta^{(k)}=-A(\delta^{(k)})r(\delta^{(k)})$$

变为：

$$(A(\delta^{(k)})A(\delta^{(k)})^{\mathrm{T}}+\mu I)\delta^{(k)}=-A(\delta^{(k)})r(\delta^{(k)})$$

对于给定的 $\mu>0$，如果 $\delta^{(k)}$ 为上式的解，则 $\delta^{(k)}$ 是下列信赖域问题的全局最优解：

$$\min\ q_k(\delta)=\frac{1}{2}\|A^{(k)\mathrm{T}}\delta+r^{(k)}\|^2$$
$$\text{s.t. } \|\delta\|\leqslant\|\delta^{(k)}\|$$

莱温伯格-马奎特法利用信赖域求解下降方向。

8.2 lsqnonneg 函数(求解非负约束的最小二乘问题)

lsqnonneg 函数的计算模型为:

$$\min_x \frac{1}{2}\|Cx-d\|_2^2$$

$$\text{s. t.} \quad x \geq 0$$

lsqnonneg 函数使用的算法为非线性最优化的单纯形法(Nelder-Mead Simplex)。

函数语法:

```
x = lsqnonneg(C,d)
x = lsqnonneg(C,d,x0)
x = lsqnonneg(C,d,x0,options)
[x,resnorm] = lsqnonneg(…)
[x,resnorm,residual] = lsqnonneg(…)
[x,resnorm,residual,exitflag] = lsqnonneg(…)
[x,resnorm,residual,exitflag,output] = lsqnonneg(…)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonneg(…)
```

函数输入:

C,d:拟合的样本数据;C 的每一行代表一组观察数据,C 的行数要大于等于 C 的列数

x0:初始迭代点

options:函数参数设置

函数输出:

X:最优点(或者迭代停止点)

Resnorm:残差平方和 norm(C * x-d)^2

Residual:残差 C * x-d 向量

Exitflag:算法退出表示

1 算法成功收敛

0 算法达到最大迭代次数

Output:算法迭代计算信息

Lambda:拉格朗日乘子

函数示例:

非负线性回归模型不含常数项示例,M 文件 Showlsqnonneg. m

```
C = [
      0.0372      0.2869
      0.6861      0.7071
      0.6233      0.6245
      0.6344      0.6170];
d = [
      0.8587
      0.1781
      0.0747
      0.8405];
[x,resnorm,residual,exitflag,output,lambda] = lsqnonneg(C,d)
```

计算结果：

```
x =
          0
     0.6929
resnorm = 0.8315
residual =
     0.6599
    -0.3119
    -0.3580
     0.4130
exitflag = 1
output =
      iterations: 1
      algorithm: 'active-set using svd'
      message: 'Optimization terminated.'
lambda =
    -0.1506
    -0.0000
```

若该问题没有非负约束，使用线性最小二乘公式：

$$Cx = d \Rightarrow x = C^{-1}d$$

```
C = [
      0.0372      0.2869
      0.6861      0.7071
```

```
      0.6233      0.6245
      0.6344      0.6170];
d=[
      0.8587
      0.1781
      0.0747
      0.8405];
X=C\d (d左除C)
```

计算结果:

```
x =
   -2.5627
    3.1108
```

8.3 lsqlin 函数(求解带约束的线性最小二乘问题)

lsqlin 函数的计算模型为:

$$\min_x \frac{1}{2}\|\boldsymbol{Cx}-\boldsymbol{d}\|_2^2$$

$$\text{s.t.}\quad \boldsymbol{Ax}\leqslant \boldsymbol{b}$$

$$\boldsymbol{Aeq}\,\boldsymbol{x}=\boldsymbol{beq}$$

$$\boldsymbol{lb}\leqslant \boldsymbol{x}\leqslant \boldsymbol{ub}$$

lsqlin 函数使用的算法为信赖域算法。

函数语法:

```
x=lsqlin(C,d,A,b)
x=lsqlin(C,d,A,b,Aeq,beq)
x=lsqlin(C,d,A,b,Aeq,beq,lb,ub)
x=lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0)
x=lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0,options)
[x,resnorm]=lsqlin(…)
[x,resnorm,residual]=lsqlin(…)
[x,resnorm,residual,exitflag]=lsqlin(…)
[x,resnorm,residual,exitflag,output]=lsqlin(…)
[x,resnorm,residual,exitflag,output,lambda]=lsqlin(…)
```

函数输入：

C,d：拟合的样本数据；C 的每一行代表一组观察数据，C 的行数要大于等于 C 的列数

A：线性不等式约束系数矩阵

B：线性不等式约束常向量

Aeq：线性等式约束系数矩阵

Beq：线性等式约束常向量

Lb：变量下界

Ub：变量上界

x0：初始迭代点

options：函数参数设置

函数输出：

X：最优点(或者迭代停止点)

Resnorm：残差平方和 norm(C * x-d)^2

Residual：残差 C * x-d 向量

Exitflag：算法退出表示

1　算法成功收敛

3　残差的变化小于算法误差控制参数

0　算法达到最大迭代次数

-2　问题无解

-4　出现变态条件，算法无法继续进行

-7　下降方向范数过小，不能进行有效迭代

Output：算法迭代计算信息

Lambda：拉格朗日乘子

函数调用示例：

下面我们将列举两个具体的使用示例，一个线性回归模型不含常数项，另一个线性回归模型含有常数项：

$$y = a_1x_1 + \cdots + a_nx_n$$

$$y = a_0 + a_1x_1 + \cdots + a_nx_n$$

其中 a_0 为常数项。

8.3.1　函数示例(1)

$$\min_x f(x) = \sum_{i=1}^{m} [(x_1c_{i,1} + \cdots + x_mc_{m,1}) - d_i]^2$$

$$\text{s.t.}\begin{pmatrix}0.2027 & 0.2721 & 0.7467 & 0.4659\\ 0.1987 & 0.1988 & 0.4450 & 0.4186\\ 0.6037 & 0.0152 & 0.9318 & 0.88462\end{pmatrix}x\leqslant\begin{pmatrix}0.5251\\ 0.2026\\ 0.6721\end{pmatrix}$$

$$(-0.1,-0.1,-0.1,-0.1)^{T}\leqslant x\leqslant(2,2,2,2)^{T}$$

线性回归模型不含常数项示例，M 文件 ShowLsqlin1.m

```
%数据矩阵 C
C=[
    0.9501    0.7620    0.6153    0.4057
    0.2311    0.4564    0.7919    0.9354
    0.6068    0.0185    0.9218    0.9169
    0.4859    0.8214    0.7382    0.4102
    0.8912    0.4447    0.1762    0.8936];
%数据矩阵 d
d=[
    0.0578
    0.3528
    0.8131
    0.0098
    0.1388];
%线性不等式约束系数矩阵
A=[
    0.2027    0.2721    0.7467    0.4659
    0.1987    0.1988    0.4450    0.4186
    0.6037    0.0152    0.9318    0.8462];
%线性不等式约束常数矩阵
b=[
    0.5251
    0.2026
    0.6721];
%变量可行下届
lb = -0.1*ones(4,1);
%变量可行上届
ub=2*ones(4,1);
[x,resnorm,residual,exitflag,output,lambda]=lsqlin(C,d,A,b,[ ], [ ], lb, ub)
```

计算结果：

```
ptimization terminated.
x =
    -0.1000
    -0.1000
     0.2152
     0.3502
resnorm =0.1672
residual =
     0.0455
     0.0764
    -0.3562
     0.1620
     0.0784
exitflag =1
output =
        iterations: 4
        algorithm: 'medium-scale: active-set'
        firstorderopt: [ ]
        cgiterations: [ ]
        message: 'Optimization terminated. '
lambda =
        lower: [4x1 double]
        upper: [4x1 double]
        eqlin: [0x1 double]
        ineqlin: [3x1 double]
```

8.3.2　函数示例(2)

$$\min_{x} f(\boldsymbol{x}) = \sum_{i=1}^{m} [(x_0 + x_1 c_{i,1} + \cdots + x_m c_{m,1}) - d_i]^2$$

$$\text{s. t.} \begin{pmatrix} 0.0 & 0.2027 & 0.2721 & 0.7467 & 0.4659 \\ 0.0 & 0.1987 & 0.1988 & 0.4450 & 0.4186 \\ 0.0 & 0.6037 & 0.0152 & 0.9318 & 0.88462 \end{pmatrix} \boldsymbol{x} \leqslant \begin{pmatrix} 0.5251 \\ 0.2026 \\ 0.6721 \end{pmatrix}$$

$$(-0.1, -0.1, -0.1, -0.1, 0.1)^{\mathrm{T}} \leqslant \boldsymbol{x} \leqslant (2,2,2,2,2)^{\mathrm{T}}$$

线性回归模型不含常数项示例，M 文件 ShowLsqlin2. m

```
% 数据矩阵 C
C = [
    1 0.9501    0.7620    0.6153    0.4057
    1 0.2311    0.4564    0.7919    0.9354
    1 0.6068    0.0185    0.9218    0.9169
    1 0.4859    0.8214    0.7382    0.4102
    1 0.8912    0.4447    0.1762    0.8936];
% 数据矩阵 d
d = [
    0.0578
    0.3528
    0.8131
    0.0098
    0.1388];
% 线性不等式约束系数矩阵
A = [
    0.0    0.2027    0.2721    0.7467    0.4659
    0.0    0.1987    0.1988    0.4450    0.4186
    0.0    0.6037    0.0152    0.9318    0.8462];
% 线性不等式约束常数矩阵
b = [
    0.5251
    0.2026
    0.6721];
% 变量可行下界
lb = -0.1 * ones(5,1);
% 变量可行上界
ub = 2 * ones(5,1);
[x,resnorm,residual,exitflag,output,lambda] = lsqlin(C,d,A,b,[ ], [ ], lb, ub)
```

计算结果：

```
Optimization terminated.
x =
   -0.0013 (常数项)
   -0.1000
```

```
    -0.1000
     0.2143
     0.3512
resnorm =0.1672
residual =
     0.0440
     0.0753
    -0.3575
     0.1604
     0.0778
exitflag =1
output =
     iterations: 6
     algorithm: 'medium-scale: active-set'
     firstorderopt: [ ]
     cgiterations: [ ]
     message: 'Optimization terminated. '
lambda =
       lower: [5x1 double]
       upper: [5x1 double]
       eqlin: [0x1 double]
       ineqlin: [3x1 double]
```

8.4 lsqnonlin 函数(求解非线性最小二乘问题)

lsqonlin 函数的计算模型为：

$$\min_{x} f(\boldsymbol{x}) = f_1^2(\boldsymbol{x}) + f_2^2(\boldsymbol{x}) + \cdots + f_n^2(\boldsymbol{x})$$

$$\text{s.t.}\quad \boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$$

lsqonlin 函数使用的算法为信赖域算法。

函数语法：

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,options)
[x,resnorm] = lsqnonlin(…)
```

```
   -0.1000
    0.2143
    0.3512
resnorm = 0.1672
residual =
    0.0440
    0.0753
   -0.3575
    0.1604
    0.0778
exitflag = 1
output =
    iterations: 6
    algorithm: 'medium-scale: active-set'
    firstorderopt: [ ]
    cgiterations: [ ]
    message: 'Optimization terminated. '
lambda =
      lower: [5x1 double]
      upper: [5x1 double]
      eqlin: [0x1 double]
      ineqlin: [3x1 double]
```

8.4 lsqnonlin 函数(求解非线性最小二乘问题)

lsqonlin 函数的计算模型为:

$$\min_{x} f(\boldsymbol{x}) = f_1^2(\boldsymbol{x}) + f_2^2(\boldsymbol{x}) + \cdots + f_n^2(\boldsymbol{x})$$

$$\text{s.t.} \quad \boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$$

lsqonlin 函数使用的算法为信赖域算法。

函数语法:

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,options)
[x,resnorm] = lsqnonlin(…)
```

```
[x,resnorm] = lsqnonlin(@lsqonlinmyfun,x0)
```

计算结果：

```
Optimization terminated: norm of the current step is less
than OPTIONS. TolX.
x =0.2578      0.2578
resnorm = 124.3622
```

8.5 lsqcurvefit 函数（求解非线性数据拟合问题）

lsqcurvefit 函数的计算模型为：

$$\min_{x}\frac{1}{2}\|F(x,xdata)-ydata\|_2^2=\frac{1}{2}\sum_{i=1}^{m}(F(x,xdata)-ydata)^2$$

lsqcurvefit 函数计算使用信赖域算法。

函数语法：

```
x = lsqcurvefit(fun,x0,xdata,ydata)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options)
[x,resnorm] = lsqcurvefit(…)
[x,resnorm,residual] = lsqcurvefit(…)
[x,resnorm,residual,exitflag] = lsqcurvefit(…)
[x,resnorm,residual,exitflag,output] = lsqcurvefit(…)
[x,resnorm,residual,exitflag,output,lambda] = lsqcurvefit(…)
[x,resnorm,residual,exitflag,output,lambda,jacobian] = lsqcurvefit(…)
```

函数输入：

Fun：优化目标函数

Xdata：Ydata：样本数据

Lb：变量下界

Ub：变量上界

x0：初始迭代点

options：函数参数设置

函数输出：

X：最优点（或者迭代停止点）

Resnorm：残差平方和 norm(C * x-d)^2

Residual：残差 C * x-d 向量

Exitflag：算法退出表示

1　算法收敛

2　x 变化小于函数误差控制参数

3　残差变化小于函数误差控制参数

4　下降方向范数小于函数误差控制参数

0　算法迭代次数达到算法的最大迭代次数

-1　算法因为输出函数终止

-2　问题无解

-4　算法无法继续进行

Output：算法迭代计算信息

Lambda：拉格朗日乘子

函数调用示例：

回归模型：$ydata(i) = x(1)e^{(x(2) xdata(i))}$

优化问题：$\min_x \frac{1}{2}\sum_{i=1}^{m}(F(x, xdata(i)) - ydata(i))^2$

编写目标函数 M 文件 lsqcurvefitmyfun. m

```
function f = lsqcurvefitmyfun(x,xdata)
F = x(1) * exp(x(2) * xdata)
```

调用 lsqcurvefit 函数 Showlsqcurvefit. m

```
% Assume you determined xdata and ydata experimentally
xdata = [0.9 1.5 13.8 19.8 24.1 28.2 35.2 60.3 74.6 81.3];
ydata = [455.2 428.6 124.1 67.3 43.2 28.1 13.1 -0.4 -1.3 -1.5];
x0 = [100, -1] ;     %初始迭代点
[x, resnorm, residual, exitflag, output, lambda] = lsqcurvefit (@ lsqcurvefitmyfun, x0,
xdata,ydata)
```

计算结果：

```
Optimization terminated: relative function value
changing by less than OPTIONS. TolFun.
x =498.8309    -0.1013
resnorm =9.5049
residual =
  Columns 1 through 9
  0.1817    -0.0610    -0.7628    -0.1196   0.2659   0.5979   1.0261
  1.5124   1.5615
```

```
  Column 10
    1.6327
exitflag = 3
output =
    firstorderopt: 0.0114
    iterations: 27
    funcCount: 84
    cgiterations: 23
    algorithm: 'large-scale: trust-region reflective Newton'
    message: [1x87 char]
lambda =
    lower: [2x1 double]
    upper: [2x1 double]
```

结果图像如图 8-1 所示。

图 8-1

图像的 matlab 程序：PlotShowlsqcuverfit. m

```
xdata = [0.9 1.5 13.8 19.8 24.1 28.2 35.2 60.3 74.6 81.3];
ydata = [455.2 428.6 124.1 67.3 43.2 28.1 13.1 -0.4 -1.3 -1.5];
x = [498.8309    -0.1013];
F = x(1) * exp(x(2) * xdata);
plot(xdata,ydata,'*');
hold on
plot(xdata,F);
```

第 9 章　0-1 整数规划

前面各章中，求解问题的变量大多数都是连续变量，但在许多实际问题中往往要求变量取值为整数。如涉及人数、飞机数、车辆数、机器台数等。本章主要讨论变量为 0 或 1 的整数规划问题，该类问题可以简称为 0-1 整数规划。

9.1　0-1 整数规划的基本模型

例如资本预算问题，决策者要对若干潜在的投资方案作出选择，决定取舍(1:取,0:舍)。设共有 n 个投资方案，$c_j(j=1,2,\cdots,n)$ 为第 j 个方案的投资收益，整个投资过程共分为 m 个阶段，b_i 为第 i 个阶段的投资总量，a_{ij} 为第 i 个阶段第 j 个方案的所需资金。目标是在各个阶段资金限制条件下使得整个投资方案的总收益最大。

这类问题是典型的决策问题。设决策变量 x_j 为对第 j 个方案的取舍(1:取,0 舍)，可得到如下整数规划问题（是 0-1 规划）：

$$\max z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s. t.} \ \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i, \ i=1,2,\cdots,m$$

$$x_j = 0, \text{或} 1, \ j=1,2,\cdots,n$$

其中约束：

$$\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i, \ i=1,2,\cdots,m$$

反映了第 i 个时期资金增长量的平衡。这里 a_{ij} 为第 i 个阶段第 j 个方案资金净流量：

（1）$a_{ij}>0$，表示第 i 个阶段第 j 个方案需要附加资金。

（2）$a_{ij}<0$，表示第 i 个阶段第 j 个方案产生资金。

右端项 b_i 为第 i 个阶段外源资金流量的增长量：

（1）$b_i>0$，表示第 i 个阶段附加资金的数量。

（2）$b_i<0$，表示第 i 个阶段抽回资金的数量。

0-1 规划的一般标准模型为：

$$\min f(x)=\sum_{j=1}^{n}c_jx_j$$

$$\text{s.t.}\ \sum_{j=1}^{n}a_{ij}x_j\leqslant b_i,\ i=1,2,\cdots,m$$

$$\sum_{j=1}^{n}e_{ij}x_j=b_i,\ i=m+1,m+2,\cdots,n$$

$$x_j=0,\ \text{或}\ 1,\ j=1,2,\cdots,n$$

用矩阵的标准形式可以表示为：

$$\min f(\boldsymbol{x})=\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$$

$$\text{s.t.}\ \boldsymbol{Ax}\leqslant\boldsymbol{b}$$

$$\boldsymbol{Aeqx}=\boldsymbol{beq}$$

$$x_j=0,\ \text{或}\ 1,\ j=1,\ 2,\ \cdots,\ n$$

式中，$\boldsymbol{A}$ 为不等式约束的系数矩阵；$\boldsymbol{b}$ 为不等式约束的常数项；$\boldsymbol{Aeq}$ 为等式约束的系数矩阵，$\boldsymbol{beq}$ 为等式约束的常数项。

9.2　分枝定界法与隐枚举法

1. 分枝定界法

分枝定界法（branch and bound）是一种求解整数规划问题的最常用算法。这种方法不但可以求解纯整数规划，还可以求解混合整数规划问题。分枝定界法也是求解 0-1 规划的主要算法。

分枝定界法的步骤为：

第 1 步：放宽或取消原问题的某些约束条件，如求整数解的条件。如果这时求出的最优解是原问题的可行解，那么这个解就是原问题的最优解，计算结束。否则这个解的目标函数值是原问题的最优解的上界（求极大值时）。

第 2 步：将放宽了某些约束条件的替代问题分成若干子问题，要求各子问题的解集合的并集要包含原问题的所有可行解，然后对每个子问题求最优解。这些子问题的最优解中的最优者若是原问题的可行解，则它就是原问题的最优解，计算结束。否则它的目标函数值就是原问题的一个新的上界。另外，各子问题的最

优解中，若有原问题的可行解的，选这些可行解的最大目标函数值，它就是原问题最优解的一个下界。

第 3 步：对最优解的目标函数值已小于这个下界的问题，其可行解中必无原问题的最优解，可以放弃。对最优解的目标函数值大于这个下界的子问题，都先保留下来，进入第 4 步。

第 4 步：在保留下的所有子问题中，选出最优解的目标函数值最大的一个，重复第 1 步和第 2 步。如果已经找到该子问题的最优可行解，那么用其目标函数值与前面保留的其他问题在内的所有子问题的可行解中目标函数值最大者，将它作为新的下界，重复第 3 步，直到求出最优解。

2. 隐枚举法

由于 0-1 规划模型的特殊性，对 0-1 规划的求解存在更简便的方法，称为“隐枚举法”（implicit enumeration method）。用分枝定界法求解整数规划时，替代问题是放宽变量的整数约束；而用隐枚举法求解 0-1 型整数规划时，替代问题是在保持变量 0-1 约束的条件下，放松问题的资源约束。下面通过例子说明隐枚举法的具体步骤。

$$\max z = 100x_1 + 30x_2 + 40x_3 + 45x_4$$

$$\text{s. t. } 50x_1 + 30x_2 + 25x_3 + 10x_4 \leqslant 95$$

$$7x_1 + 2x_2 + x_3 + 4x_4 \leqslant 11$$

$$2x_1 + x_2 + x_3 + 10x_4 \leqslant 12$$

$$x_4 \leqslant x_2 + x_3$$

$$x_1,\ x_2,\ x_3,\ x_4 = 0 \text{ 或 } 1$$

将上式标准化，为使得系数为正，设 $y_i = 1 - x_i$，$i = 1, 2, 3, 4$，$f = 215 - z$，取得的新的 0-1 规划为：

$$\min f = 100y_1 + 30y_2 + 40y_3 + 45y_4$$

$$\text{s. t. } -50y_1 - 30y_2 - 25y_3 - 10y_4 \leqslant -20$$

$$-7y_1 - 2y_2 - y_3 - 4y_4 \leqslant -4$$

$$-2y_1 - y_2 - y_3 - 10y_4 \leqslant -2$$

$$y_2 + y_3 - y_4 \leqslant 1$$

$$y_1,\ y_2,\ y_3,\ y_4 = 0 \text{ 或 } 1$$

下面用二元枚举树说明计算过程（见图 9-1）。

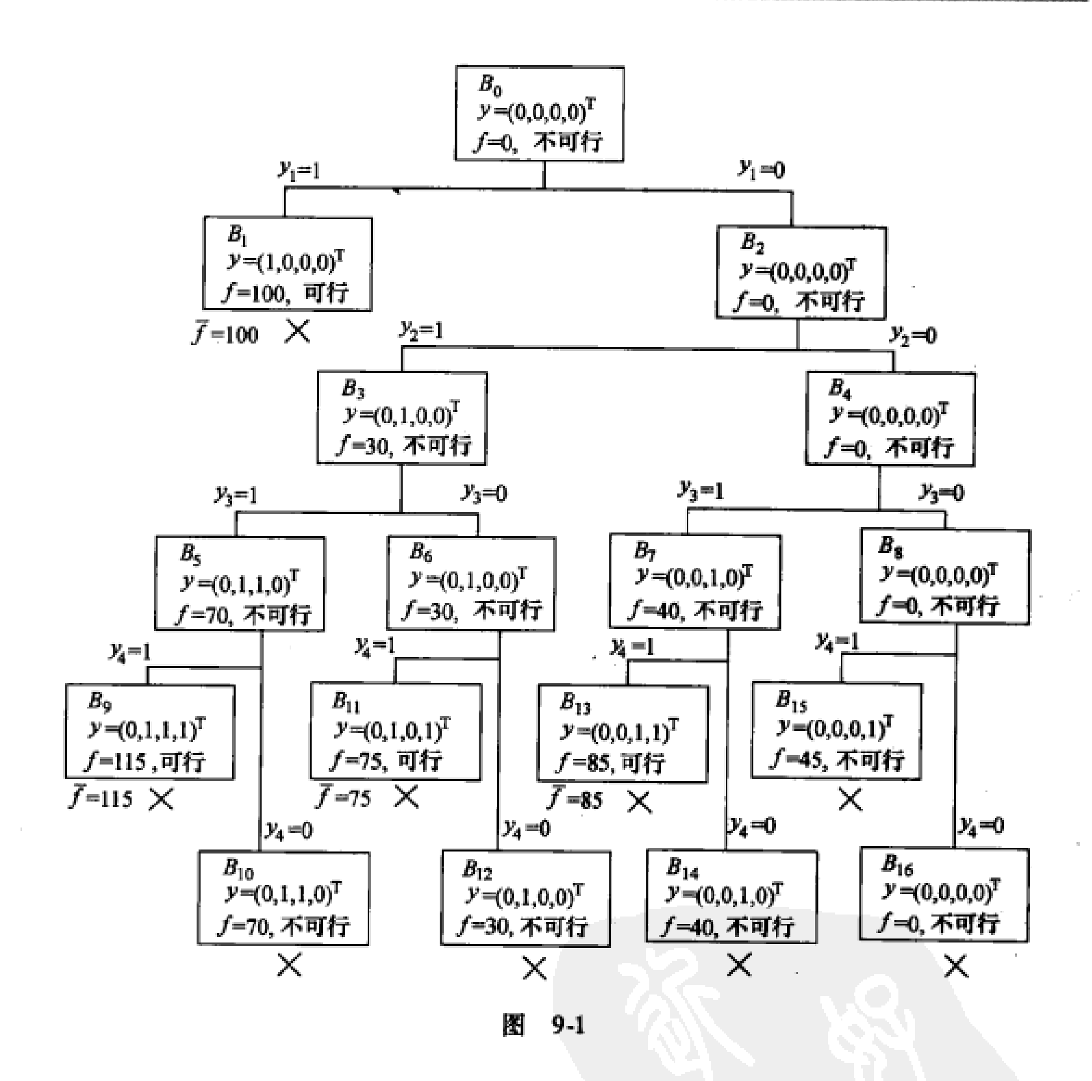


图　9-1

9.3　bintprog 函数(求解 0-1 整数规划)

bintprog 函数是 MATLAB 主要的求解 0-1 规划的函数，该函数使用的是分枝定界法算法。函数求解问题的标准模型为：

$$\min \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}$$

$$\text{s. t. } \boldsymbol{Ax} \leqslant \boldsymbol{b}$$

$$\boldsymbol{Aeqx} = \boldsymbol{beq}$$

$$x_j = 0\text{；或 } 1,\ j = 1,\ 2,\ \cdots,\ n$$

函数语法：

x = bintprog(f)

```
x = bintprog(f,A,b)
x = bintprog(f,A,b,Aeq,beq)
x = bintprog(f,A,b,Aeq,beq,x0)
x = bintprog(f,A,b,Aeq,Beq,x0,options)
[x,fval] = bintprog(…)
[x,fval,exitflag] = bintprog(…)
[x,fval,exitflag,output] = bintprog(…)
```

函数输入：

F：目标函数系数向量

A：线性不等式约束的系数矩阵

B：线性不等式约束的常数向量

Aeq：线性等式约束的系数矩阵

Beq：线性等式约束的常数向量

X0：算法初始迭代点

Options：算法参数设置

函数输出：

X：最优点(迭代停止点)

Fval：最优点(迭代停止点)对应的函数值

Exitflag：算法结束信息

1：算法收敛

0：算法达到最大迭代次数

-2：目标问题无解

-4：算法搜索节点达到最大节点数

-5：算法达到最大的迭代时间

-6：算法求解 LP(线性规划)次数达到最大次数

Output：函数迭代信息

9.3.1 函数示例(1)

$$\min f = 100y_1 + 30y_2 + 40y_3 + 45y_4$$

$$\text{s.t.} \quad -50y_1 - 30y_2 - 25y_3 - 10y_4 \leqslant -20$$

$$-7y_1 - 2y_2 - y_3 - 4y_4 \leqslant -4$$

$$-2y_1 - y_2 - y_3 - 10y_4 \leqslant -2$$

$$y_2 + y_3 - y_4 \leqslant 1$$

$$y_1,\ y_2,\ y_3,\ y_4 = 0 \text{ 或 } 1$$

编写 M 文件 test91.m

```
options = optimset('display','iter');
% 显示迭代点
f = [100,30,40,45];
% 目标函数系数
A = [-50   -30   -25   -10;
      -7    -2    -1    -4;
      -2    -1    -1   -10;
       0     1     1    -1];
% 不等式约束的系数矩阵
b = [-20; -4; -2; 1];
% 不等式约束的常数向量
[x,fval,exitflag,output] = bintprog(f,A,b,[], [], [], options)
```

计算结果：

Explored nodes	Obj of LP relaxation	Obj of best integer point	Unexplored nodes	Best lower bound on obj	Relative gap between bounds
1	48	-	2	48	-
* 6	75	75	3	52.5	30%
11	65	75	0	65	13%

```
Optimization terminated.
x =
     0
     1
     0
     1
fval = 75
exitflag = 1
output =
          iterations: 21    (迭代次数)
               nodes: 11    (搜索节点数)
                time: 0.0300(计算时间)
           algorithm: ' LP-based branch-and-bound'
                      (算法:基于线性规划的分枝定界法)
      branchStrategy: ' maximum integer infeasibility'
```

(分枝策略,maximum integer infeasibility)

nodeSrchStrategy:' best node search' (节点搜索策略)

message:' Optimization terminated. '

9.3.2 函数示例(2)

将示例(1)中的 $y_2+y_3-y_4\leqslant 1$ 不等式约束，变为等式约束 $y_1+y_2+y_3+y_4=3$。

$$\min f=100y_1+30y_2+40y_3+45y_4$$

$$\text{s. t.}\begin{cases}-50y_1-30y_2-25y_3-10y_4\leqslant -20\\ -7y_1-2y_2-y_3-4y_4\leqslant -4\\ -2y_1-y_2-y_3-10y_4\leqslant -2\\ y_1+y_2+y_3+y_4=3\\ y_1,\ y_2,\ y_3,\ y_4=0\text{ 或 }1\end{cases}$$

编写 M 文件 test92. m

```
options = optimset('display','iter');
% 显示迭代点
f = [100,30,40,45 ];
% 目标函数系数
A = [ -50  -30  -25  -10;
      -7   -2   -1   -4;
      -2   -1   -1  -10]
% 不等式约束的系数矩阵
b = [ -20; -4; -2];
% 不等式约束的常数向量
Aeq = [1,1,1,1];
% 等式约束的系数矩阵
beq = 3;
% 等式约束的常数项
[x,fval,exitflag,output] = bintprog(f,A,b,Aeq,beq,[], options)
```

计算结果:

Explored nodes	Obj of LP relaxation	Obj of best integer point	Unexplored nodes	Best lower bound on obj	Relative gap between bounds
* 1	115	115	0	115	0%

```
Optimization terminated.
x =
      0
      1
      1
      1
fval = 115
exitflag = 1
output =
              iterations: 2
                   nodes: 1
                    time: 0.2303
               algorithm: 'LP-based branch-and-bound'
          branchStrategy: 'maximum integer infeasibility'
        nodeSrchStrategy: 'best node search'
                 message: 'Optimization terminated.'
```

9.4 分派问题

在现实世界，经常会遇到这样的问题：有 n 个人恰好可承担 n 项任务，一项任务由一个人完成，一个人只完成一项任务；由于每个人的专长不同，完成各项任务的效率也就不同，于是产生了应指派哪个人去完成哪项任务，才能使完成 n 项任务总的效率最高(所需总时间最少)的问题。这类问题称为指派问题或分派问题(assignment problem)。

9.4.1 指派问题的数学模型

对应每个指派问题都有一个已知的效率矩阵，其元素 $c_{ij}\geqslant 0(i,j=1,2,\cdots,n)$ 表示第 i 个人完成第 j 项任务的效率(如时间或成本等)。引入 0-1 变量 x_{ij}：

$$x_{ij}=\begin{cases}1 & \text{指派第 } i \text{ 人去完成第 } j \text{ 项任务}\\ 0 & \text{不指派第 } i \text{ 人去完成第 } j \text{ 项任务}\end{cases}$$

于是问题的数学模型是：

$$\min z=\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}x_{ij}$$

$$\begin{cases}\sum_{j=1}^{n} x_{ij}=1 & (i=1,2,\cdots,n)\\ \sum_{i=1}^{n} x_{ij}=1 & (j=1,2,\cdots,n)\\ x_{ij}=0 \text{ 或 } 1\end{cases}$$

第一个约束条件说明第 i 个人只能完成一项任务，第二个约束条件说明第 j 项任务只能由一个人去完成。从指派问题的数学模型可以看出，指派问题是 0-1 规划的特例，也是运输问题的特例。既然如此，指派问题自然可以用整数规划或运输问题的求解方法去求解；然而，利用指派问题的特点有更简便的求解方法。

分派问题是 0-1 规划问题，匈牙利法是解决分派问题的一种有效方法。1955 年库恩(kuln)利用匈牙利数学家克尼格(Konig)的关于矩阵中独立"0"元素的定理，提出了求解指派问题的一种方法，习惯上称之为匈牙利法。匈牙利法的基本原理就是"效率矩阵的任一行(或列)减去(或加上)任一常数，指派问题的最优解不会受到影响"。

9.4.2 分派问题的转换及 AssignProb 函数

为便于 MATLAB 的 0-1 规划 Bintprog 函数求解，我们可以将分派问题变为标准的 0-1 规划问题。可以将矩阵变量 $\boldsymbol{x}$，按行拉成向量变量 $\boldsymbol{xv}$。将矩阵系数 $\boldsymbol{c}$，按行拉成向量系数 $\boldsymbol{cv}$。

$$\boldsymbol{xv}=(x_{n(i-1)+j}),i,j=1,2,\cdots,n$$

$$\boldsymbol{cv}=(c_{n(i-1)+j}),i,j=1,2,\cdots,n$$

$$\min z=\sum_{i=1}^{n\times n} cv_i xv_i$$

$$\text{s.t.}\ \sum_{j=1}^{n} xv_{j+n(i-1)}=1,\ i=1,\ 2,\ \cdots,\ n$$

$$\sum_{i=1}^{n} xv_{j+n(i-1)}=1,\ j=1,\ 2,\ \cdots,\ n$$

$$xv_i=0 \text{ 或 } 1$$

其中，$\sum_{j=1}^{n} xv_{j+n(i-1)}=1$，$i=1,2,\cdots,n$，每行只能有一个元素为 1，即每个人只能完成一项任务；

$\sum_{i=1}^{n} xv_{j+n(i-1)}=1$，$j=1,2,\cdots,n$，每列只能有一个元素为 1，即每项任务只能由一个人来完成。

根据上述算法我们编写了 AssignProb，便于读者使用 AssignProb 间接调用 bintprog 函数求解分派问题。

AssignProb 函数代码(见 AssignProb. m)：

```
function [AP,fval,exitflag,output] = AssignProb(C)
% code by ariszheng
% email: ariszheng@gmail.com
% this function is to solve Assignment Problem
% Input:
%     C is the AP matrix, the rowNum of C is the WorkerNum, and the colNum
%     of C is workNum
% ouput:
%     AP: is the Assignment Problem Matrix
[rowNum,ColNum] = size(C);
if rowNum~ = ColNum
    error('the Assignment Problem Matrix error');
end
f = reshape(C',1,rowNum * ColNum);
AeqRow = zeros(rowNum,rowNum * ColNum);
for i = 1:rowNum
    for j = 1:ColNum
        AeqRow(i,ColNum * (i-1) +j) =1;
    end
end
AeqCol = zeros(ColNum,rowNum * ColNum);
for j = 1:ColNum
    for i = 1:rowNum
        AeqCol(j,ColNum * (i-1) +j) =1;
    end
end
Aeq = [AeqRow;
    AeqCol]
beq = ones(1,rowNum + ColNum)
[x,fval,exitflag,output] = bintprog(f,[], [], Aeq, beq);
AP = reshape(x,rowNum,ColNum)';
```

函数语法：

[AP,fval,exitflag,output] = AssignProb(C)

函数输入：

C：分派系数矩阵(输入的分派矩阵必须为方阵)

函数输出：

AP：最优分派矩阵

Fval：最优分派矩阵对应的函数值

Exitflag：算法结束信息

1：算法收敛

0：算法达到最大迭代次数

-2：目标问题无解

-4：算法搜索节点达到最大节点数

-5：算法达到最大的迭代时间

-6：算法求解 LP(线性规划)次数达到最大次数

Output：函数迭代信息

9.4.3 AssignProb 函数示例(1)

例如：某装修公司有甲、乙、丙、丁、戊 5 个装修队。现在公司接到 5 项装修任务 A、B、C、D、E。根据经验，各个装修队完成不同的任务所需要的时间如表 9-1 所示。

表 9-1

	A	B	C	D	E
甲	32	17	34	36	25
乙	21	31	21	22	19
丙	24	29	40	28	39
丁	26	35	41	33	29
戊	33	27	31	42	22

如何分派可以使得总用时最少?

对以上分派问题建立模型：

$$\min z = \sum_{i=1}^{5} \sum_{j=1}^{5} c_{ij} x_{ij}$$

$$\text{s. t.} \sum_{j=1}^{n} x_{ij} = 1,\ j = 1, 2, \cdots, 5$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ i = 1, 2, \cdots, 5$$

$$x_{ij} = 0 \text{ 或 } 1$$

其中约束的意义为可行解矩阵，每行只能有一个元素为 1，即每个人只能完成一项任务；每列只能有一个元素为 1，即每项任务只能由一个人来完成。

分派系数矩阵为：
$$c = \begin{pmatrix} 32 & 17 & 34 & 36 & 25 \\ 21 & 31 & 21 & 22 & 19 \\ 24 & 29 & 40 & 28 & 39 \\ 26 & 35 & 41 & 33 & 29 \\ 33 & 27 & 31 & 42 & 22 \end{pmatrix}$$

编程 Matlab 程序见 testAP1. m

```
C = [32,17,34,36,25;
     21,31,21,22,19;
     24,29,40,28,29;
     26,35,41,33,29;
     33,27,31,42,22]
[AP,fval,exitflag,output] = AssignProb(C)
```

计算结果：

```
Optimization terminated.
AP =
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     1     0     0     0     0
     0     0     0     0     1
fval = 114
exitflag = 1
output =
          iterations: 35
               nodes: 1
```

```
                time: 0.0901
           algorithm: ' LP-based branch-and-bound'
      branchStrategy: ' maximum integer infeasibility'
     nodeSrchStrategy: ' best node search'
             message: ' Optimization terminated. '
```

9.4.4 AssignProb 函数示例(2)

例如，由甲、乙、丙、丁 4 个人完成 5 项任务 A、B、C、D、E。根据经验，每个人完成任务的时间如表 9-2 所示。

表 9-2

	A	B	C	D	E
甲	5	9	11	22	17
乙	24	23	11	5	18
丙	14	7	8	20	12
丁	4	22	16	3	25

如何分派可以使得总用时最少？

虚设 1 人为戊，把戊完成 A、B、C、D、E 任务所需时间设为所有人完成该任务用时最少的那个人所用时间，系数矩阵如表 9-3 所示。

表 9-3

	A	B	C	D	E
甲	5	9	11	22	17
乙	24	23	11	5	18
丙	14	7	8	20	12
丁	4	22	16	3	25
戊	4	7	8	3	12

编程 Matlab 程序见 testAP2. m

```
C=[5,9,11,22,17;
    24,23,11,5,18;
    14,7,8,20,12;
    4,22,16,3,25;
    4,7,8,3,12]
```

```
[AP,fval,exitflag,output] = AssignProb(C)
```

计算结果：

```
Optimization terminated.
AP =
     1     0     0     0     0
     0     0     1     0     0
     0     0     0     0     1
     0     0     0     1     0
     0     1     0     0     0
fval =38
exitflag =1
output =
          iterations: 12
               nodes: 1
                time: 0.0300
           algorithm: 'LP-based branch-and-bound'
      branchStrategy: 'maximum integer infeasibility'
     nodeSrchStrategy: 'best node search'
             message: 'Optimization terminated.'
```

分派方案如表 9-4 所示。

表　9-4

	A	B	C	D	E
甲	1				
乙			1		
丙		1			1
丁				1	

9.4.5　AssignProb 函数示例(3)

上述问题(2)为人数小于任务数，即有一人完成多项任务，可以通过虚设人，增加人数的方法将其变成标准的分派问题；同样，当任务数小于人数，即有人空闲，可以通过虚设任务的方法将其变成标准的分派问题。

例如，由甲、乙、丙、丁，戊 5 个人完成 4 项任务 A，B，C，D。根据经验，每个人完成任务的时间如表 9-5 所示。

表 9-5

	甲	乙	丙	丁	戊
A	10	2	3	15	9
B	5	10	15	2	4
C	15	5	14	7	15
D	20	15	13	6	8

另外，由于某种原因，甲必须分到一项任务，丁不能承担任务 D，求满足这些条件如何分派可以使得总用时最少？

解：虚设任务 E，甲不能轮空，丁不能承担任务 D，得到新的系数矩阵，如表 9-6 所示。

表 9-6

	甲	乙	丙	丁	戊
A	10	2	3	15	9
B	5	10	15	2	4
C	15	5	14	7	15
D	20	15	13	1000	8
E	1000	0	0	0	0

注释：(甲,E)、(丁,D)本应为无穷大，由于要用计算机计算，所以给定一个相对其他数较大的数据即可。

编程 Matlab 程序见 testAP3. m

```
C=[10,2,3,15,9;
   5,10,15,2,4;
   15,5,14,7,15;
   20,15,13,1000,8;
   1000,0,0,0,0]
[AP,fval,exitflag,output]=AssignProb(C)
```

计算结果：

```
Optimization terminated.
AP =
     0     0     1     0     0
     1     0     0     0     0
```

```
      0    1    0    0    0
      0    0    0    0    1
      0    0    0    1    0
fval = 21
exitflag = 1
output =
          iterations: 21
               nodes: 1
                time: 0.0200
           algorithm: 'LP-based branch-and-bound'
      branchStrategy: 'maximum integer infeasibility'
   nodeSrchStrategy: 'best node search'
             message: 'Optimization terminated. '
```

分配方案如表 9-7 所示。

表　9-7

	甲	乙	丙	丁	戊
A			1		
B	1				
C		1			1
D				1	

丁轮空，最小用时为 21。

第10章　目 标 规 划

在科学研究、经济建设和生产实践中，人们经常遇到一类含有多个目标的数学规划问题，我们称之为多目标规划。本章介绍一种特殊的多目标规划，叫目标规划(Goal Programming)，这是美国学者 Charnes 等在1952年提出来的。目标规划在实践中的应用十分广泛，它的重要特点是对各个目标分级加权与逐级优化，这符合人们处理问题要分轻重缓急、保证重点的思考方式。

本章分目标规划模型、fgoalattain 函数两个部分进行介绍。其中 fgoalattain 函数是 MATLAB 求解的主要函数。

10.1　目标规划模型

10.1.1　问题提出

为了便于理解目标规划数学模型的特征及建模思路，我们首先通过一个简单的例子来说明。

例：某公司分厂用一条生产线生产两种产品 A 和 B，每周生产线运行时间为60h，生产一台 A 产品需要4h，生产一台 B 产品需要6h。根据市场预测，A、B 产品平均销售量分别为每周9台、8台，它们的销售利润分别为12万元、18万元。在制定生产计划时，经理考虑下述4项目标：首先，产量不能超过市场预测的销售量；其次，工人加班时间最少；第三，希望总利润最大；最后，要尽可能满足市场需求，当不能满足时，市场认为 B 产品的重要性是 A 产品的2倍。试建立这个问题的数学模型。

讨论：

若把总利润最大看作目标，而把产量不能超过市场预测的销售量、工人加班时间最少和要尽可能满足市场需求的目标看作约束，则可建立一个单目标线性规划模型。

设决策变量 x_1，x_2 分别为产品 A，B 的产量，则

$$\max Z = 12x_1 + 18x_2$$

$$\text{s.t. } 4x_1 + 6x_2 \leqslant 60$$

$$x_1 \leqslant 9$$

$$x_2 \leqslant 8$$
$$x_1,\ x_2 \geqslant 0$$

容易求得上述线性规划的最优解为$(9,4)^T$到$(3,8)^T$所在线段上的点，最优目标值为$Z^* = 180$万元，即可选方案有多种。

在实际上，这个结果并不完全符合决策者的要求，它只实现了经理的第一、二、三条目标，而没有达到最后一个目标。进一步分析可知，要实现全体目标是不可能的。

下面我们结合上例介绍目标规划模型。

10.1.2　目标规划模型的基本概念

我们把上例中的 4 个目标表示为不等式，仍设决策变量x_1，x_2分别为产品 A，B 的产量。那么，

第一个目标为：$x_1 \leqslant 9$，$x_2 \leqslant 8$；

第二个目标为：$4x_1 + 6x_2 \leqslant 60$；

第三个目标为：希望总利润最大，要表示成不等式需要找到一个目标上界，这里可以估计为 252($12 \times 9 + 18 \times 8$)，于是有$12x_1 + 18x_2 \geqslant 252$；

第四个目标为：$x_1 \geqslant 9$，$x_2 \geqslant 8$。

下面引入与建立目标规划数学模型有关的概念。

（1）正、负偏差变量d^+，d^-。我们用正偏差变量d^+表示决策值超过目标值的部分；负偏差变量d^-表示决策值不足目标值的部分。因决策值不可能既超过目标值同时又未达到目标值，故恒有$d^+ \times d^- = 0$。

（2）绝对约束和目标约束。我们把所有等式、不等式约束分为两部分：绝对约束和目标约束。

绝对约束是指必须严格满足的等式约束和不等式约束。如在线性规划问题中考虑的约束条件，不能满足这些约束条件的解称为非可行解，所以它们是硬约束。设举例中生产 A，B 产品所需原材料数量有限制，并且无法从其他渠道予以补充，则构成绝对约束。

目标约束是目标规划特有的，我们可以把约束右端项看作要努力追求的目标值，但允许发生正负偏差，在约束中加入正、负偏差变量来表示，于是称它们是软约束。

对于上例，我们有如下目标约束：

①$x_1 + d_1^- - d_1^+ = 9$

②$x_2 + d_2^- - d_2^+ = 8$

③$4x_1 + 6x_2 + d_3^- - d_3^+ = 60$

④$12x_1 + 18x_2 + d_4^- - d_4^+ = 252$

(3) 优先因子与权系数。对于多目标问题，设有 L 个目标函数 f_1，f_2，…，f_L，决策者在要求达到这些目标时，一般有主次之分。为此，我们引入优先因子 P_i，$i=1$，2，…，L。不妨设预期的目标函数优先顺序为 f_1，f_2，…，f_L，我们把要求第一位达到的目标赋于优先因子 P_1，次位的目标赋于优先因子 P_2，…，并规定 $P_i \gg P_{i+1}$，$i=1$，2，…，$L-1$。即在计算过程中，首先保证 P_1 级目标的实现，这时可不考虑次级目标；而 P_2 级目标是在实现 P_1 级目标的基础上考虑的，依此类推。当需要区别具有相同优先因子的若干个目标的差别时，可分别赋于它们不同的权系数 w_j。优先因子及权系数的值均由决策者按具体情况来确定。

(4) 目标规划的目标函效。目标规划的目标函数是通过各目标约束的正、负偏差变量和赋于相应的优先等级来构造的。决策者的要求是尽可能从某个方向缩小偏离目标的数值。于是，目标规划的目标函数应该是求极小：$\min f=f(d^+, d^-)$。其基本形式有以下三种：

1) 要求恰好达到目标值，即使相应目标约束的正、负偏差变量都要尽可能地小。这时取 $\min(d^+ + d^-)$。

2) 要求不超过目标值，即使相应目标约束的正偏差变量要尽可能地小。这时取 $\min(d^+)$。

3) 要求不低于目标值，即使相应目标约束的负偏差变量要尽可能地小。这时取 $\min(d^-)$。

对于上例，我们根据决策者的考虑知：

第一优先级要求　$\min(d_1^+ + d_2^+)$。

第二优先级要求　$\min(d_3^+)$。

第三优先级要求　$\min(d_4^-)$。

第四优先级要求 $\min(d_1^- + 2d_2^-)$。这里，当不能满足市场需求时，市场认为 B 产品的重要性是 A 产品的 2 倍，即减少 B 产品其影响是 A 产品的 2 倍，因此我们引入了 2:1 的权系数。

综合上述分析，我们可得到下列目标规划模型：

$$
\begin{aligned}
\min f = & P_1(d_1^+ + d_2^+) + P_2 d_3^+ + P_3 d_4^- + P_4(d_1^- + 2d_2^-) \\
\text{s.t. } & x_1 + d_1^- - d_1^+ = 9 \\
& x_2 + d_2^- - d_2^+ = 8 \\
& 4x_1 + 6x_2 + d_3^- - d_3^+ = 60 \\
& 12x_1 + 18x_2 + d_4^- - d_4^+ = 252 \\
& x_1,\ x_2,\ d_i^-,\ d_i^+ \geqslant 0,\ i=1,2,3,4
\end{aligned}
$$

10.1.3 目标规划模型的一般形式

根据上面讨论，我们可以得到目标规划的一般形式如下：

$$(\text{LGP})\begin{cases}\min \sum_{l=1}^{L} P_l\left[\sum_{k=1}^{K}(w_{lk}^- d_k^- + w_{lk}^+ d_k^+)\right] \\ \text{s.t. } \sum_{j=1}^{n} c_{kj}x_j + d_k^- - d_k^+ = g_k,\ k=1,2,\cdots,K \\ \sum_{j=1}^{n} a_{ij}x_j = (\leqslant,\geqslant) b_i,\ i=1,2,\cdots,m \\ x_j,\ d_k^-,\ d_k^+ \geqslant 0,\ j=1,2,\cdots,n; k=1,2,\cdots,K \end{cases}$$

(LGP)中的第二行是 K 个目标约束，第三行是 m 个绝对约束，c_{kj}和 g_k 是目标参数。

10.1.4 利用 linprog 函数求解目标规划

这里以 10.1.1 中的例子作为程序示例：

$$\min f = P_1(d_1^+ + d_2^+) + P_2 d_3^+ + P_3 d_4^- + P_4(d_1^- + 2d_2^-)$$

$$\begin{aligned} \text{s.t. } & x_1 + d_1^- - d_1^+ = 9 \\ & x_2 + d_2^- - d_2^+ = 8 \\ & 4x_1 + 6x_2 + d_3^- - d_3^+ = 60 \\ & 12x_1 + 18x_2 + d_4^- - d_4^+ = 252 \\ & x_1,\ x_2,\ d_i^-,\ d_i^+ \geqslant 0,\ i=1,\ 2,\ 3,\ 4 \end{aligned}$$

为了便于编程我们将

$$\begin{gathered} d_1^- = x_3,\ d_1^+ = x_4, \\ d_2^- = x_5,\ d_2^+ = x_6, \\ d_3^- = x_7,\ d_3^+ = x_8 \\ d_4^- = x_9,\ d_4^+ = x_{10} \\ x_i \geqslant 0,\ i=3,\ 4,\ \cdots,\ 10 \end{gathered}$$

原问题变为标准的约束优化问题：

$$\min f = P_1(x_4 + x_6) + P_2 x_8 + P_3 x_9 + P_4(x_3 + 2x_5)$$

$$\begin{aligned} \text{s.t. } & x_1 + x_3 - x_4 = 9 \\ & x_2 + x_5 - x_6 = 8 \\ & 4x_1 + 6x_2 + x_7 - x_8 = 60 \\ & 12x_1 + 18x_2 + x_9 - x_{10} = 252 \end{aligned}$$

$$x_i \geq 0,\ i=1,\ 2,\ \cdots,\ 10$$

利用 linprog 函数的求解步骤为：

设 P = [1000,100,10,1]

$$\begin{aligned} f &= P_1(x_4+x_6)+P_2x_8+P_3x_9+P_4(x_3+2x_5) \\ &= P_4x_3+P_1x_4+2P_4x_5+P_1x_6+P_2x_8+P_3x_9 \\ &= x_3+1000x_4+2x_5+1000x_6+100x_8+10x_9 \end{aligned}$$

调用 linprog 函数求解 solvefgoalMyfun1. m

```
% solvefgoalMyfun1
F = [0,0,1,1000,2,1000,0,100,10,0]
% 等式约束系数矩阵
Aeq = [1,0,1, -1,0,0,0,0,0,0;
        0,1,0,0,1, -1,0,0,0,0;
        4,6,0,0,0,0,1, -1,0,0;
        12,18,0,0,0,0,0,0,1, -1];
% 等式约束常数向量
beq = [9;8;60;252];
% 可行解下界
lb = zeros(1,10);
[x,fval,exitflag,output] = linprog(F,[], [], Aeq, beq, lb, [])
```

计算结果：

```
Optimization terminated.
x =
    3.0000
    8.0000
    6.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
   72.0000
    0.0000
fval = 726.0000
exitflag = 1
```

```
output =
        iterations: 7
        algorithm: 'large-scale: interior point'
        cgiterations: 0
            message: 'Optimization terminated. '
```

最优解为 x = [3,8]

如果目标函数或者约束函数中含有非线性方程，则可以用求解非线性约束问题的 fmincon 函数对其进行求解。

10.2　fgoalattain 函数

MATLAB 提供一种内置函数 fgoalattain，为求解目标规划问题，但其目标规划模型相对简单。对每个目标只引入一个松弛变量 γ，而且没有涉及同优先级式不同变量的权重问题。

fgoalattain 函数计算模型为：

$$\min_{x,\gamma} \gamma$$

$$\text{s.t.} \quad \begin{aligned} & \boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{weight}\gamma \leqslant \boldsymbol{goal} \\ & \boldsymbol{c}(\boldsymbol{x}) \leqslant \boldsymbol{0} \\ & \boldsymbol{ceq}(\boldsymbol{x}) = \boldsymbol{0} \\ & \boldsymbol{Ax} \leqslant \boldsymbol{b} \\ & \boldsymbol{Aeqx} = \boldsymbol{beq} \\ & \boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub} \end{aligned}$$

这里 $\boldsymbol{x}$，$\boldsymbol{b}$，$\boldsymbol{beq}$，$\boldsymbol{lb}$，$\boldsymbol{ub}$，$\boldsymbol{weight}$ 为向量，$\boldsymbol{A}$ 与 $\boldsymbol{Aeq}$ 为矩阵，$\boldsymbol{F}(\boldsymbol{x})$ 为目标函数，$\boldsymbol{c}(\boldsymbol{x})$，$\boldsymbol{ceq}(\boldsymbol{x})$ 为非线性约束，$\boldsymbol{Ax} \leqslant \boldsymbol{b}$，$\boldsymbol{Aeqx} = \boldsymbol{beq}$ 为线性约束，$\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$ 为可行解的区间约束。

多目标优化同时涉及一系列对象。fgoalattain 函数求解该问题的基本算法是目标达到法。该方法为目标函数建立起目标值。多目标优化的具体算法在前面进行了更详细的介绍。在本次实现过程中，使用了松弛变量 γ 作为模糊变量，同时最小化目标向量 $\boldsymbol{F}(\boldsymbol{x})$；$\boldsymbol{goal}$ 参数是一系列目标达到值。在进行优化之前，通常不知道对象是否会达到目标。使用权向量 $\boldsymbol{weight}$ 可以控制是没有达到还是溢出。fgoalattain 函数使用序列二次规划法(SQP)，算法中对于一维搜索和 Hessian 矩阵进行了修改。在一维搜索中，对精确目标函数，当有一个目标函数不再发生改善时，一维搜索终止。修改的 Hessian 矩阵借助于本问题的结构，也采用 attainfactor 参数，包含解出的 γ 值。γ 取负值时表示目标溢出。

函数语法：

x = fgoalattain(fun,x0,goal,weight)

x = fgoalattain(fun,x0,goal,weight,A,b)

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq)

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub)

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub,nonlcon)

x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub,nonlcon,… options)

[x,fval] = fgoalattain(…)

[x,fval,attainfactor] = fgoalattain(…)

[x,fval,attainfactor,exitflag] = fgoalattain(…)

[x,fval,attainfactor,exitflag,output] = fgoalattain(…)

[x,fval,attainfactor,exitflag,output,lambda] = fgoalattain(…)

函数输入：

Fun：目标函数

X0：初始迭代点

Goal：目标希望达到的向量值

Weight：目标权系数

A：线性不等式约束系数矩阵

B：线性不等式约束的常数向量

Aeq：线性等式约束系数矩阵

Beq：线性等式约束的常数向量

lb：可行区域下界

Ub：可行区域上界

Nonlcon：非线性约束

Options：优化参数设置：

优化参数选项。你可以用 optimset 函数设置或改变这些参数的值，具体可以参看附录：MATLAB 优化工具箱参数设置。

函数输出：

X：最优点(或者结束迭代点)

Fval：最优值(或者结束迭代点对应的函数值)

Attainfactor：attainfactor 变量是超过或低于目标的个数。若 attainfactor 为负，则目标已经溢出；若 attainfactor 为正，则目标个数还未达到。

Exitflag：迭代停止标识

1：算法收敛停止

4：搜索方向的范数小于给定的误差控制参数
5：梯度（导数）范数小于给定的误差控制参数
0：算法达到最大迭代次数
-1：算法由 output function 终止
-2：未找到可行解
Output ：算法输出（算法计算信息等）
Ambda：最优点（或者结束迭代点）拉格朗日乘子

10.2.1　函数示例(1)

某化工厂拟生产两种新产品 A 和 B，其生产设备费用分别为：A，2 万元/t；B，5 万元/t。这两种产品均将造成环境污染，设由公害所造成的损失可折算为：A，4 万元/t；B，1 万元/t。由于条件限制，工厂生产产品 A 和 B 的最大生产能力各为每月 5t 和 6t，而市场需要这两种产品的总量每月不少于 7t。试问工厂如何安排生产计划，在满足市场需要的前提下，使设备投资和公害损失均达到最小。

该工厂决策认为，这两个目标中环境污染应优先考虑，设备投资的目标值为 20 万元，公害损失的目标为 12 万元。

设工厂每月生产产品 A 为 x_1，B 为 x_2，设备投资费为 $f_1(x)$，公害损失费为 $f_2(x)$，则这个问题可表达为多目标优化问题：

$$
\begin{aligned}
&\min \gamma \\
&\text{s. t. } 2x_1+5x_2-w_1\gamma=20 \\
&\qquad 4x_1+x_2-w_2\gamma=12 \\
&\qquad x_1\leqslant 5 \\
&\qquad x_2\leqslant 6 \\
&\qquad x_1+x_2\geqslant 7 \\
&\qquad x_1,\ x_2\geqslant 0
\end{aligned}
$$

（1）编写目标函数的 M 文件 fgoalmyfun2. m

```
function f = fgoalmyfun2(x)
f(1) =2 * x(1) +5 * x(2);
f(2) =4 * x(1) +x(2);
```

（2）调用 fgoalattain 函数 solveFgoal2. m

```
% 给定目标，权重按目标比例确定，给出初值
goal = [20 12];
weight = [20 12];
```

```
x0 = [2 5];
%给出约束条件的系数
A = [1 0;0 1; -1 -1];
b = [5 6 -7];
lb = zeros(2,1);
[x, fval, attainfactor, exitflag] = fgoalattain(@fgoalmyfun2, x0, goal, weight,
A, b, [], [], lb, [])
```

计算结果：

```
Optimization terminated: magnitude of search direction less than 2 * options. TolX
and maximum constraint violation is less than options. TolCon.
Active inequalities (to within options. TolCon = 1e-006):
  lower    upper    ineqlin   ineqnonlin
                       3          1
                                  2
x = 2.9167    4.0833
fval = 26.2500    15.7500
attainfactor = 0.3125
exitflag = 4
```

故工厂每月生产产品 A 为 2.9167t，B 为 4.0833t。设备投资费和公害损失费的目标值分别为 26.2500 万元和 15.7500 万元。达到因子为 0.3125，计算收敛。

10.2.2 函数示例(2)

某厂生产两种产品 A 和 B，已知生产 A 产品 100kg 需 8 个工时，生产 B 产品 100kg 需 10 个工时。假定每日可用的工时数为 40，且希望不雇临时工，也不加班生产。这两种产品每 100kg 均可获利 100 元。此外，有个顾客要求每日供应他 B 种产品 600kg。问应如何安排生产计划？

设生产 A，B 两种产品的数量分别为 x_1 和 x_2（均以 100kg 计），为了使生产计划比较合理，要求用人尽量少，获利尽可能多，另外 B 种产品的产量尽量多。由题意建立下面的数学模型：

$$
\begin{aligned}
&\min \gamma \\
\text{s.t.}\ & 8x_1 + 10x_2 - w_1\gamma \leqslant 40 \\
& -100x_1 - 100x_2 - w_2\gamma \leqslant -800 \\
& x_2 - w_3\gamma \leqslant 6 \\
& 8x_1 + 10x_2 \leqslant 40
\end{aligned}
$$

$$x_2 \geqslant 6$$

$$x_1,\ x_2 \geqslant 0$$

（1）首先需要编写目标函数的 M 文件 fgoalmyfun3. m，返回目标计算值

```
function f = myfun(x)
f(1) =8 * x(1) +10 * x(2);
f(2) = -100 * x(1) -100 * x(2);
f(3) = -x(2);
```

（2）调用 fgoalattain 函数 solveFgoal3. m

```
%给定目标，权重按目标比例确定，给出初值
goal =[40  -800  -6];
weight =[40  -800  -6];
x0 =[2 2];
%给出约束条件的系数
    A =[8 10;0  -1];
    b =[40; -6];
lb = zeros(2,1);
% 设置函数评价的最大次数为 20000 次
options = optimset('MaxFunEvals',20000);
[x,fval,attainfactor,exitflag] = fgoalattain(@ fgoalmyfun3,x0,goal,weight,
A,b,[], [], lb, [], [], options);
```

计算结果为：

```
x =2.0429      1.9458
fval =35.8007  -398.8648     -1.9458
attainfactor = -0.0646
exitflag =0
```

经过 5000 次迭代以后，得生产 A，B 两种产品的数量各为 204.29kg 和 194.58kg。

第 11 章　最大最小问题

最大最小(minimax)问题是多目标规划的一种，其多个目标的量纲相同可以进行比较，最大最小的意义使得多个目标值中最大的那个目标值最小化。例如，如果存在关系：

$$f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n$$

可以得到：

$$\min\{\max(f_1, f_2, \cdots, f_n)\} \Leftrightarrow \min f_n$$

11.1　最大最小问题模型

$$\min\{\max(f_1,\ f_2,\ \cdots,\ f_n)\}$$
$$\text{s.t. } h_i(\boldsymbol{x}) = 0,\ i = 1,\ 2,\ \cdots,\ m$$
$$g_j(\boldsymbol{x}) \leqslant 0,\ j = 1,\ 2,\ \cdots,\ n$$

其中 f_1，f_2，…，f_n 为多个目标的目标函数，$h_i(\boldsymbol{x})$ 为等式约束(线性或非线性等式约束)，$g_j(\boldsymbol{x})$ 为不等式约束(线性或非线性不等式约束)。

最小最大模型变为最大最小模型的转换方式：

$$\max\{\min(f_1, f_2, \cdots, f_n)\}$$
$$\Rightarrow \min\{-\min(f_1, f_2, \cdots, f_n)\}$$
$$\Rightarrow \min\{\max(-f_1, -f_2, \cdots, -f_n)\}$$

例：计划在 A，B，C，D，E 五个城市之间建造一个垃圾处理厂 P（见图11-1），

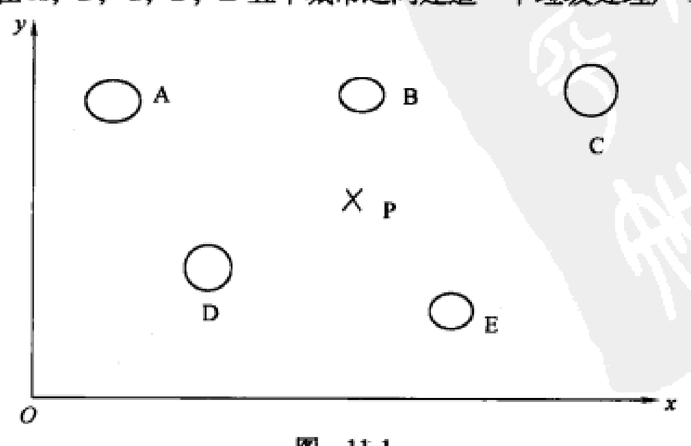


图　11-1

为了使五个城市将垃圾运往垃圾处理厂 P 的运输成本尽可能相差不大。由于运输成本主要由城市与垃圾处理厂之间的距离决定，所以垃圾处理厂 P 的选址目标是使得五个城市到垃圾处理厂 P 的距离尽量相近。城市坐标如表 11-1 所示。

表 11-1

城　市	x 坐标	y 坐标
A	1.5	6.8
B	6.0	7.0
C	8.9	6.9
D	3.5	4.0
E	7.4	3.1

解：由平面距离公式可得：

$$f_1 = d_{A\to P} = \sqrt{(x-1.5)^2+(y-6.8)^2}$$
$$f_2 = d_{B\to P} = \sqrt{(x-6.0)^2+(y-7.0)^2}$$
$$f_3 = d_{C\to P} = \sqrt{(x-8.9)^2+(y-6.9)^2}$$
$$f_4 = d_{D\to P} = \sqrt{(x-3.5)^2+(y-4.0)^2}$$
$$f_5 = d_{E\to P} = \sqrt{(x-7.4)^2+(y-3.1)^2}$$

目标函数为：

$$\min\{\max(f_1, f_2, \cdots, f_5)\}$$

11.2　fminimax 函数

fminimax 函数计算模型：

$$\min_{x}\{\max(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \cdots, f_n(\boldsymbol{x}))\}$$
$$\text{s.t.}\ \boldsymbol{c}(\boldsymbol{x}) \leqslant \boldsymbol{0}$$
$$\boldsymbol{ceq}(\boldsymbol{x}) \leqslant \boldsymbol{0}$$
$$\boldsymbol{Ax} \leqslant \boldsymbol{b}$$
$$\boldsymbol{Aeqx} = \boldsymbol{beq}$$
$$\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$$

这里 $\boldsymbol{x}$，$\boldsymbol{b}$，$\boldsymbol{beq}$，$\boldsymbol{lb}$，$\boldsymbol{ub}$ 为向量，$\boldsymbol{A}$ 与 $\boldsymbol{Aeq}$ 为矩阵，$f(\boldsymbol{x})$ 为目标函数，$\boldsymbol{c}(\boldsymbol{x})$，$\boldsymbol{ceq}(\boldsymbol{x})$ 为非线性约束，$\boldsymbol{Ax} \leqslant \boldsymbol{b}$，$\boldsymbol{Aeqx} = \boldsymbol{beq}$ 为线性约束，$\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$ 为可行解的区间约束。

fminimax 函数使用的约束优化算法都是目前比较普适的有效算法，主要使用序列二次规划(Sequential Quadratic Programming,SQP)算法。由于这些算法都具有一定的复杂性，具体算法这里不再详述。

函数语法：

x = fminimax(fun,x0)

x = fminimax(fun,x0,A,b)

x = fminimax(fun,x,A,b,Aeq,beq)

x = fminimax(fun,x,A,b,Aeq,beq,lb,ub)

x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)

x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)

[x,fval] = fminimax(…)

[x,fval,maxfval] = fminimax(…)

[x,fval,maxfval,exitflag] = fminimax(…)

[x,fval,maxfval,exitflag,output] = fminimax(…)

[x,fval,maxfval,exitflag,output,lambda] = fminimax(…)

函数输入：

Fun：目标函数

X0：初始迭代点

A：线性不等式约束系数矩阵

B：线性不等式约束的常数向量

Aeq：线性等约束系数矩阵

Beq：线性等式约束的常数向量

lb：可行区域下界

Ub：可行区域上界

Nonlcon：非线性约束

Options：优化参数设置

函数输出：

X：最优点(或者结束迭代点)

Fval：最优值(或者结束迭代点对应的函数值)

Maxfval：最大算数值

Exitflag：迭代停止标识

1：算法收敛停止

4：搜索方向的范数小于给定的误差控制参数

5：梯度(导数)范数小于给定的误差控制参数

0：算法达到最大迭代次数
-1：算法由 output function 终止
-2：未找到可行解
Output ：算法输出（算法计算信息等）
Ambda：最优点（或者结束迭代点）拉格朗日乘子

11.2.1　函数示例（1）

求解最大最小问题

$$\min\{\max(f_1, f_2, \cdots, f_5)\}$$

$$f_1 = d_{A \to P} = \sqrt{(x-1.5)^2 + (y-6.8)^2}$$

$$f_2 = d_{B \to P} = \sqrt{(x-6.0)^2 + (y-7.0)^2}$$

$$f_3 = d_{C \to P} = \sqrt{(x-8.9)^2 + (y-6.9)^2}$$

$$f_4 = d_{D \to P} = \sqrt{(x-3.5)^2 + (y-4.0)^2}$$

$$f_5 = d_{E \to P} = \sqrt{(x-7.4)^2 + (y-3.1)^2}$$

编写目标函数 minimaxMyfun. m

```
function f = minimaxMyfun(x)
f = zeros(1,5);
f(1) = sqrt((x(1) -1.5)^2 + (x(2) -6.8)^2);
f(2) = sqrt((x(1) -6.0)^2 + (x(2) -7.0)^2);
f(3) = sqrt((x(1) -8.9)^2 + (x(2) -6.9)^2);
f(4) = sqrt((x(1) -3.5)^2 + (x(2) -4.0)^2);
f(5) = sqrt((x(1) -7.4)^2 + (x(2) -3.1)^2);
```

调用 fminimax 函数 showfminimax1. m

```
options = optimset('display','iter');
% 显示迭代过程
x0 = [0.0; 0.0];
% 初始迭代点
[x,fval,maxfval,exitflag,output,lambda] = fminimax(@minimaxMyfun,x0)
```

计算结果：

```
Optimization terminated: magnitude of search direction less than 2 * options.TolX
and maximum constraint violation is less than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):
```

```
  lower       upper      ineqlin    ineqnonlin
                                        1
                                        3
                                        5
x =
     5.2093
     6.1608
fval =3.7640      1.1530      3.7640      2.7551      3.7640
maxfval =3.7640
exitflag =4
output =
       iterations: 7
       funcCount: 56
       lssteplength: 1
       stepsize: 3.9107e-009
       algorithm: ' minimax SQP, Quasi-Newton, line_search'
       firstorderopt: [ ]
              message: [1x142 char]
lambda =
            lower: [2x1 double]
            upper: [2x1 double]
            eqlin: [0x1 double]
         eqnonlin: [0x1 double]
          ineqlin: [0x1 double]
ineqnonlin: [0x1 double]
```

最佳垃圾处理厂的建设位置坐标为(5.2093,6.1608)

11.2.2 函数示例(2)

原示例问题变为：

计划在 A，B，C，D，E 五个城市之间建造一个垃圾处理厂 P，为了使五个城市将垃圾运往垃圾处理厂 P 的运输成本尽可能相差不大。由于运输成本主要由城市与垃圾处理厂之间的距离决定，所以垃圾处理厂 P 的选址目标是使得五个城市到垃圾处理厂 P 的距离尽量相近。

另外，A，B，C，D，E 五个城市之间有一条高速公路，为了便于垃圾处理

厂的垃圾转运方便，垃圾处理厂要在该公路边上建造，如图 11-2 所示。该公路在图上的坐标线为：$y=x-2.5$。城市坐标如表 11-2 所示。

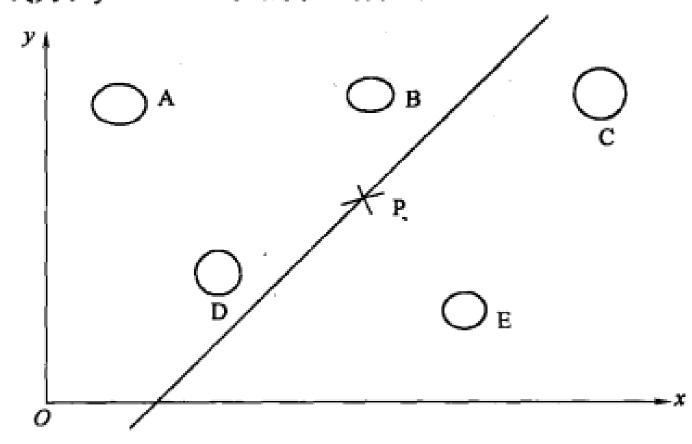


图　11-2

表　11-2

城　　市	x 坐标	y 坐标
A	1.5	6.8
B	6.0	7.0
C	8.9	6.9
D	3.5	4.0
E	7.4	3.1

该问题的方程为：

$$\min\{\max(f_1, f_2, \cdots, f_5)\}$$
$$\text{s.t.}\quad x_1 - x_2 = 2.5$$
$$f_1 = d_{A\to P} = \sqrt{(x-1.5)^2+(y-6.8)^2}$$
$$f_2 = d_{B\to P} = \sqrt{(x-6.0)^2+(y-7.0)^2}$$
$$f_3 = d_{C\to P} = \sqrt{(x-8.9)^2+(y-6.9)^2}$$
$$f_4 = d_{D\to P} = \sqrt{(x-3.5)^2+(y-4.0)^2}$$
$$f_5 = d_{E\to P} = \sqrt{(x-7.4)^2+(y-3.1)^2}$$

编写目标函数 minimaxMyfun. m

```
function f = minimaxMyfun(x)
f = zeros(1,5);
```

```
f(1) = sqrt((x(1) -1.5)^2 + (x(2) -6.8)^2);
f(2) = sqrt((x(1) -6.0)^2 + (x(2) -7.0)^2);
f(3) = sqrt((x(1) -8.9)^2 + (x(2) -6.9)^2);
f(4) = sqrt((x(1) -3.5)^2 + (x(2) -4.0)^2);
f(5) = sqrt((x(1) -7.4)^2 + (x(2) -3.1)^2);
```

调用 fminimax 函数 showfminimax1. m

```
options = optimset('display','iter');
%显示迭代过程
x0 = [0.0; 0.0];
%初始迭代点
Aeq = [1, -1];
beq = 2.5;
[x,fval,maxfval,exitflag,output,lambda] = fminimax(@minimaxMyfun,x0,[], [],
Aeq, beq)
```

计算结果:

```
Optimization terminated: magnitude of search direction less than 2 * options.TolX
and maximum constraint violation is less than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):
  lower        upper       ineqlin    ineqnonlin
                                           1
x =
    5.4000
    2.9000
fval = 5.5154     4.1437     5.3151     2.1954     2.0100
maxfval = 5.5154
exitflag = 4
output =
       iterations: 8
        funcCount: 53
     lssteplength: 1
         stepsize: 2.0670e-006
        algorithm: 'minimax SQP, Quasi-Newton, line_search'
    firstorderopt: []
          message: [1x142 char]
```

```
lambda =
        lower: [2x1 double]
        upper: [2x1 double]
        eqlin: -0.7071
     eqnonlin: [0x1 double]
      ineqlin: [0x1 double]
   ineqnonlin: [0x1 double]
```

最佳垃圾处理厂的建设位置坐标为(5.4000，2.9000)，在公路边上。

第12章　层次分析法（AHP）

在实际生活中，由于问题中含有大量的主、客观因素。许多要求与期望是模糊的，相互之间还会存在一些矛盾。所以，这类决策问题单纯依靠构造一个数学模型来求解往往行不通。面对这类决策问题，运筹学工作者进行了大量的研究，寻求不同的方法。

美国运筹学家 T. L. Saaty 教授在20世纪70年代提出的层次分析法（Analytic Hierarchy Process，AHP）是处理这类问题最有效的方法之一。以 T. L. Saaty 为首的小组曾成功把层次分析法用于电力计划、苏丹运输研究、美国高等教育事业1985～2000展望、1985年世界石油价格预测等重大研究项目上。

层次分析法的主要特征是，合理地把定性与定量的决策结合起来，按照思维心理的规律把决策过程层次化、数量化。

本章主要介绍层次分析法的基本概念与 MATLAB 求解 AHP 的方法。

12.1　层次分析法的基本概念

运用层次分析法进行决策时主要分四个步骤：

（1）建立系统的递阶层次模型。

（2）构造两两比较的判断矩阵。

（3）针对某一个标准（准则），计算各被支配元素的权重。

（4）计算当前一层元素关于总目标的排序权重。

12.1.1　建立系统的递阶层次模型

利用层次分析法解决问题，首先是建立层次结构模型。这一步必须建立在对问题及其环境充分理解、分析的基础上。作为一个工具，层次分析法模型的层次结构大体分成三类：

第一类：最高层，又称顶层、目标层。这层只有一个元素，一般是决策问题的预订目标或理想结果。

第二类：中间层，又称准则层。这一层可以有多个子层，每个子层可有多个元素，它们包括所有为实现目标所涉及的中间环节。这些环节常常考虑的是需要考虑的准则、子准则。

第三类：最底层，又称措施层、方案层。这一层的元素是为了实现目标可供选择的各种措施、决策或方案。

我们称层次分析结构模型中各项为结构模型的元素。层次分析结构模型如图 12-1 所示。

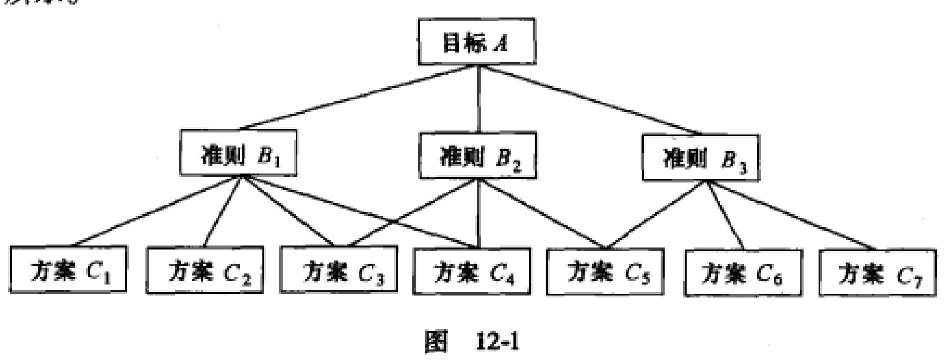


图　12-1

在实际建模过程中有以下几点需要说明：

（1）除顶层和底层之外，各元素受上层某一个或某些元素的支配，同时又支配下层的某些元素。

（2）层次之间的支配关系可以是完全的，也可以是不完全的，即某元素只支配其下层的某些元素，有时甚至是隔层支配，例如图 12-2。

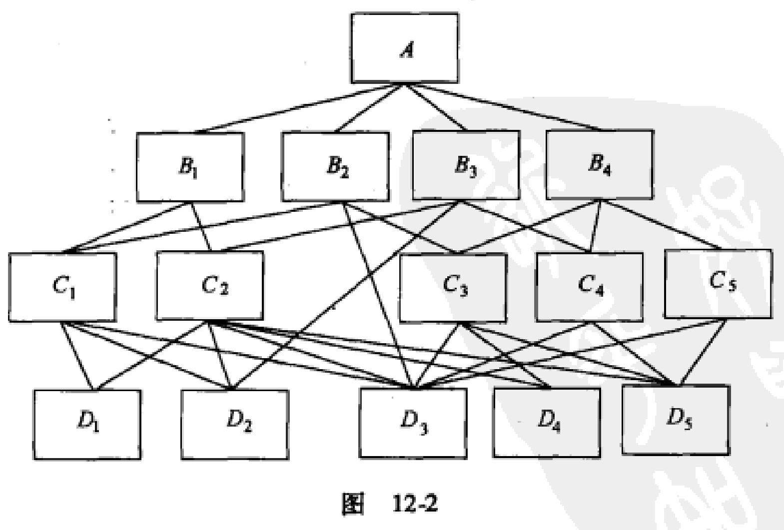


图　12-2

（3）递阶层次结构中的层数与问题的复杂程度有关，一般不受限制。

（4）为了避免判断上的困难，每个层次中的元素所支配的下层元素一般不超过 9 个。若实际问题中元素所支配的元素多于 9 个时，可将该层分成若干子层。

根据上述特征，称这种自上而下的支配关系所形成的层次结构为递阶层次结构。

12.1.2 构造判断矩阵

当一个递阶层次结构建立以后，需要确定一个上层元素 z（除底层外）所支配的下层若干元素 x_1，x_2，…，x_m 关于这个 z 的排序权重。这些权重 p_1，p_2，…，p_m 常常表示为百分数，即满足 $0 \leqslant p_i \leqslant 1$，且 $\sum_{i=1}^{m} p_i = 1$。

要直接确定这些权重，一般是很困难的，因为此类决策问题中，各被支配的元素相对于准则 z 往往只有一个定性的评价，如“好”、“差”等，所以对于多个元素排列，区分度不够。

层次分析法提出使用两两比较的方式建立判断矩阵。设上层元素 z 支配的 m 个元素为 x_1，x_2，…，x_m，对于 i，$j=1$，2，…，m，以 a_{ij}表示 x_i 与 x_j 关于 z 的影响之比，可得到矩阵：

$$\boldsymbol{A}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{pmatrix}$$

称 $\boldsymbol{A}$ 为 x_1，x_2，…，x_m 关于 z 两两比较的判断矩阵，简称判断矩阵。

为了便于操作，Saaty 建议使用 1 ~ 9 及其倒数共 17 个数作为标度来确定 a_{ij} 的值，习惯称为 9 标度法。表 12-1 为 9 标度表。

表 12-1

含义	x_i 与 x_j 同重要	x_i 比 x_j 稍重要	x_i 比 x_j 重要	x_i 比 x_j 强重要	x_i 比 x_j 极重要
a_{ij}	1	3	5	7	9
	2	4	6	8	

显然，两两比较矩阵具有下列性质：

（1）对于任意 i，$j=1$，2，…，m，有 $a_{ij}>0$。

（2）对于任意 i，$j=1$，2，…，m，有 $a_{ij}=1/a_{ji}$。

（3）对于任意 $i=1$，2，…，m，有 $a_{ii}=1$。

12.1.3 单层权重计算

层次分析法权重的计算采用的是特征根法，即采用判断矩阵最大特征根对应

的归一化特征向量为权重向量。计算方法如下：

（1）求单一准则下元素两两比较的判断矩阵 $A=(a_{ij})_{n\times n}$。

（2）求 A 的最大特征值 λ_{max} 及其对应的特征向量 $u=(u_1,u_2,\cdots,u_n)$。

（3）将 u 归一化，即

$$w_i=\frac{u_i}{\sum_{i=1}^{n}u_i},i=1,2,\cdots,n$$

为保证权重的质量，还要对判断矩阵进行一致性检验。检验方法如下：

（1）计算判断矩阵 A 的最大特征值 λ_{max}。

（2）求一致性指标 C. I. （Consistency Index）：

$$\text{C. I.}=\frac{\lambda_{max}-n}{n-1}$$

（3）查表求相应的平均随机一致性指标 R. I. （Random Index）。一致性随机指标表如表 12-2 所示。

（4）计算一致性比率 C. R. （Consistency Ratio）：

$$\text{C. R.}=\frac{\text{C. I.}}{\text{R. I.}}$$

表　12-2

矩阵阶数	3	4	5	6	7	8
R. I.	0.58	0.90	1.12	1.24	1.32	1.41
矩阵阶数	9	10	11	12	13	
R. I.	1.45	1.49	1.51	1.54	1.56	

（5）判断：当 C. R. <0.1 时，认为判断矩阵 A 有满意一致解；C. R. >0.1 时，应考虑修正判断矩阵。

12.1.4　各层元素对目标层的合成权重计算

层次分析法的最终目的是求得底层，即方案层各元素关于目标层的排序权重。计算方法如下：

设：已计算出第 $k-1$ 层 n_{k-1} 个元素对于目标的合成权重为：

$$w^{(k-1)}=(w_1^{(k-1)},w_2^{(k-1)},\cdots,w_{n_{k-1}}^{(k-1)})$$

第 k 层的 n_k 个元素关于第 $k-1$ 层第 j 个元素的单一准则排序权重向量为：

$$u_j^{(k)}=(u_{1j}^{(k)},u_{2j}^{(k)},\cdots,u_{n_k j}^{(k)})$$

$$j=1,2,\cdots,n_{k-1}$$

对应第 k 层的 n_k 个元素是完全的。当某些元素不受第 $k-1$ 层 n_{k-1} 个元素支配时，相应的位置用 0 补充。于是得到 $n_k \times n_{k-1}$ 矩阵：

$$U^{(k)} = \begin{pmatrix} u_{11}^{(k)} & u_{12}^{(k)} & \cdots & u_{1n_{k-1}}^{(k)} \\ u_{21}^{(k)} & u_{22}^{(k)} & \cdots & u_{2n_{k-1}}^{(k)} \\ \vdots & \vdots & & \vdots \\ u_{n_k 1}^{(k)} & u_{n_k 2}^{(k)} & \cdots & u_{n_k n_{k-1}}^{(k)} \end{pmatrix}$$

计算出第 k 层 n 个元素对于目标的合成权重为：

$$w^{(k)} = U^{(k)} w^{(k-1)} \Rightarrow w^{(k)} = U^{(k)} U^{(k-1)} \cdots U^{(3)} w^{(2)}$$

整体一致性检验：

C. I. 计算公式为：

$$\mathrm{C.I.}^{(k)} = (\mathrm{C.I.}_1^{(k)}, \mathrm{C.I.}_2^{(k)}, \cdots, \mathrm{C.I.}_{n_{k-1}}^{(k)}) w^{(k-1)} = \sum_{j=1}^{n_{k-1}} w_j^{(k-1)} \mathrm{C.I.}_j^{(k)}$$

R. I. 计算公式为：

$$\mathrm{R.I.}^{(k)} = (\mathrm{R.I.}_1^{(k)}, \mathrm{R.I.}_2^{(k)}, \cdots, \mathrm{R.I.}_{n_{k-1}}^{(k)}) w^{(k-1)} = \sum_{j=1}^{n_{k-1}} w_j^{(k-1)} \mathrm{R.I.}_j^{(k)}$$

一致性比率为：

$$\mathrm{C.R.}^{(k)} = \frac{\mathrm{C.I.}^{(k)}}{\mathrm{R.I.}^{(k)}}$$

12.2 函数 AHPWeightVector（单层权重计算）

层次分析法的主要计算为特征值与特征向量计算、一致性比率计算、矩阵计算等，这些都可以用 MATLAB 来实现。

12.2.1 函数说明

函数语法：

[WeightVector, lamdaMax, CR] = AHPWeightVector(DecMatrix)

函数说明：

AHPWeightVector：根据判断矩阵计算权重向量

函数输入：

DecMatrix：判断矩阵（正互反矩阵）

函数输出：
WeightVector：权重向量
lamdaMax：最大特征值
CR：　　　一致性比率
　　　　　如果判断矩阵阶数小于 3，返回 0.0
函数原码：

```
function[ WeightVector,CR ] = AHPWeightVector( DecMatrix)
% code by ariszheng 2007-11-11 ariszheng@ gmail. com
% AHPWeightVector：根据判断矩阵计算权重向量
% 函数输入：
% DecMatrix：判断矩阵（正互反矩阵）
% 函数输出：
% WeightVector：权重向量
% CR：          一致性比率
%               如果判断矩阵阶数小于 3，返回 0.0
% 判断矩阵 DecMatrix 的阶数
n = size( DecMatrix,1 );
% 计算特征向量与特征值矩阵使用 eig 函数
[ Eigenvector,Eigenvalues ] = eig( DecMatrix );
lamdaMax = Eigenvector;
% 从特征值矩阵中提取特征值，即 Eigenvector 的对角元
Evalues = diag( Eigenvalues );
% 最大特征值，及其对应的特征向量
[ maxEvalue,point ] = max( Evalues );
WeightVector = Eigenvector( :,point)/sum( Eigenvector( :,point) );
% 随机一致性指标阶数从 3 ~ 13
RI = [0.58,0.90,1.12,1.24,1.32,1.41,1.45,1.49,1.51,1.54,1.56];
if n >2
    % 当阶数大于 2 时候计算 CI、CR
    CI = ( maxEvalue - n)/( n - 1);
    CR = CI/RI( n - 2);
else
    CR = 0.0;
end
```

12.2.2 函数示例（1）

计算判断矩阵：

$$A=\begin{pmatrix}1 & 2 & 5\\1/2 & 1 & 3\\1/5 & 1/3 & 1\end{pmatrix}$$

编写 MATLAB 程序 TestAHPWeightVector1. m

```
A = [1,2,5;1/2,1,3;1/5,1/3,1];
disp('DecMatrix = A');
[WeightVector,CR] = AHPWeightVector(A)
```

计算结果：

```
DecMatrix = A
WeightVector =
    0.5816
    0.3090
    0.1095
lamdaMax =3.0037
CR =0.0032
```

权重向量为[0.5816,0.3090,0.1095]

最大特征值为 3.0037

一致性比率为 0.0032

12.2.3 函数示例（2）

计算判断矩阵：

$$A=\begin{pmatrix}1 & 2 & 5\\1/2 & 1 & 7\\1/5 & 1/7 & 1\end{pmatrix}$$

编写 MATLAB 程序 TestAHPWeightVector2. m

```
disp('DecMatrix = B');
B = [1,2,5;1/2,1,7;1/5,1/7,1];
[WeightVector,CR] = AHPWeightVector(B)
```

计算结果：

```
DecMatrix = B
WeightVector =
```

```
    0.5415
    0.3816
    0.0768
lamdaMax =3.1189
CR =0.1025
权重向量为[0.5415,0.3816,0.0768]
最大特征值为3.1189
一致性比率为0.1025 >0.1
```

12.3　函数 AHPSolver（AHP 求解函数）

AHPSolver 可以根据递阶层次模型从上到下根据 12.1.4 中公式依次计算。

12.3.1　AHPSolver 代码

```
function AHPSolver
% code by ariszheng 2007-11-11 ariszheng@gmail.com
HierarchyNum = input('Hierarchy Num of Ahp\n');
NumOfElem = input('Number of element in every Hierarchy\n');
weightKtoTop =0;
for i =1:HierarchyNum -1
      sprintf('plese input the DecMatrix of %dth Hierarchy',i)
      k =0;
      while true
          flag = input('Do you want input DecMatrix in this Hierarchy? 1 or 0 \n');
          if flag ==1
              k = k +1;
              sprintf('plese input the %d row %d th:DecMatrix',i,k)
              sprintf('the position of Ö§ÅäÔªËØ in %d th ²ã',i)
              DecMatrix(k).DecEle = input('\n');
              sprintf('the position of ±>>Ö§ÅäÔªËØ in %d ²ã',i+1)
              DecMatrix(k).SupEle = input('\n');
              DecMatrix(k).Matrix = input('DecMatirx \n');
              [DecMatrix(k).Weight, DecMatrix(k).LamdaMax, DecMatrix
              (k).CR] = AHP
```

```
            WeightVector(DecMatrix(k).Matrix);
            disp('the weight of this DecMatrix:');
            DecMatrix(k).Weight
            disp('the DecMatrix.LamdaMax:');
            DecMatrix(k).LamdaMax
            disp('DecMatrix.CR:');
            DecMatrix(k).CR
        else
            disp('this DecMatrix in this Hierarchy Input over');
            break;
        end
    end
    if i==1;
        weightKtoTop = DecMatrix(1).Weight;
        CR = DecMatrix(1).CR;
    else
        for j = 1:k
            CR = CR + weightKtoTop(j) * DecMatrix(j).CR;
        end
        U = zeros(NumOfElem(i+1),NumOfElem(i));
        for j = 1:k
            U(DecMatrix(j).SupEle,j) = DecMatrix(j).Weight;
        end
        weightKtoTop = U * weightKtoTop;
    end
    sprintf('the weight of %d Hierarchy element is',i+1);
    weightKtoTop
    end
    end
```

12.3.2 AHPSolver 使用示例

例如：某决策问题的递阶层次结构如图 12-3 所示。各判断矩阵如下，用 MATLAB 求解。

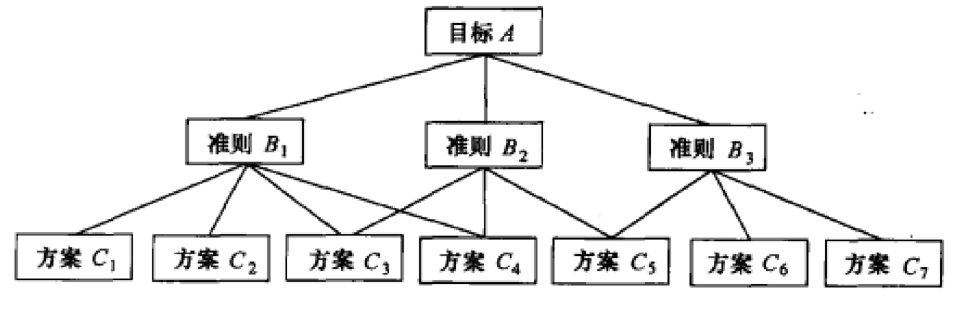


图　12-3

$$A=\begin{pmatrix}1 & 2 & 6\\ 1/3 & 1 & 2\\ 1/6 & 1/2 & 1\end{pmatrix},B_1=\begin{pmatrix}1 & 1/3 & 1/5 & 1/9\\ 3 & 1 & 1/2 & 1/7\\ 5 & 2 & 1 & 1/4\\ 9 & 7 & 1/4 & 1\end{pmatrix}$$

$$B_2=\begin{pmatrix}1 & 2 & 7\\ 1/2 & 1 & 4\\ 1/7 & 1/4 & 1\end{pmatrix},B_3=\begin{pmatrix}1 & 2 & 5\\ 1/2 & 1 & 4\\ 1/5 & 1/3 & 1\end{pmatrix}$$

程序实例：

```
>> AHPSolver(调用 AHPSolver 函数)
Hierarchy Num of Ahp(AHP 问题层数)
3
Number of element in every Hierarchy(AHP 每层元素个数)
[1,3,7]
plese input the DecMatrix of 1th Hierarchy(输入第一层判断矩阵)
Do you want input DecMatrix in this Hierarchy? 1or 0
1(继续)
plese input the 1 row 1 th:DecMatrix(支配元素在第 1 层的位置,从左向右)
the position of 支配元素 in 1 th 层
1(位置 1)
the position of 被支配元素 in 2 层
  [1,2,3](位置从左到右)
DecMatirx(判断矩阵 A)
[1,3,6;1/3,1,2;1/6,1/2,1]
the weight of this DecMatrix:(判断矩阵的权重)
    0.6667
```

```
      0.2222
      0.1111
the DecMatrix. LamdaMax:(最大特征值)
      3
DecMatrix. CR:(一致性比率)
      0
Do you want input DecMatrix in this Hierarchy? 1or 0
0(第一层终止输入)
this DecMatrix in this Hierarchy Input over
weightKtoTop =(第二层对第一层的权重)
      0.6667
      0.2222
      0.1111
plese input the DecMatrix of 2th Hierarchy
Do you want input DecMatrix in this Hierarchy? 1or 0
1(继续)
plese input the 2 row 1 th:DecMatrix
the position of 支配元素 in 2 th 层(B1 输入)
1
the position of 被支配元素 in 3 层
  [1,2,3,4](B2 的支配元素在第三层)
DecMatirx
[1,1/3,1/5,1/9;3,1,1/2,1/7;5,2,1,1/4;9,7,4,1](B1)
the weight of this DecMatrix:
      0.0478
      0.1085
      0.1998
      0.6439
the DecMatrix. LamdaMax:
      4.0891
DecMatrix. CR:
      0.0330
Do you want input DecMatrix in this Hierarchy? 1or 0
1(继续)
```

plese input the 2 row 2 th:DecMatrix
the position of 支配元素 in 2 th 层(B2 输入)
2
the position of 被支配元素 in 3 层
[3,4,5](B2 支配第三层的 C3,C4,C5)
DecMatirx
[1,2,7;1/2,1,4;1/7,1/4,1]
the weight of this DecMatrix:
    0.6026
    0.3150
    0.0823
the DecMatrix. LamdaMax:
    3.0020
DecMatrix. CR:
ans =
    0.0017
Do you want input DecMatrix in this Hierarchy? 1or 0
1(继续)
plese input the 2 row 3 th:DecMatrix
the position of 支配元素 in 2 th 层(B3)
3
the position of 被支配元素 in 3 层
 [5,6,7]
DecMatirx
[1,2,5;1/2,1,3;1/5,1/3,1](B3)
the weight of this DecMatrix:
    0.5816
    0.3090
    0.1095
the DecMatrix. LamdaMax:
    3.0037
DecMatrix. CR:
    0.0032
Do you want input DecMatrix in this Hierarchy? 1or 0

```
0(结束(
this DecMatrix in this Hierarchy Input over
weightKtoTop =(第三层关于目标层的权重)
    0.0319
    0.0724
    0.2671
    0.4993
    0.0829
    0.0343
    0.0122
```



第13章　遗传算法

遗传算法是模拟生物在自然环境中的遗传和进化过程而形成的一种自适应全局优化概率搜索算法。它最早由美国密执安大学的Holland教授提出，起源于20世纪60年代对自然和人工自适应系统的研究。20世纪70年代，De Jong基于遗传算法的思想在计算机上进行了大量的纯数值函数优化计算实验。在一系列研究工作的基础上，20世纪80年代，由Goldberg进行归纳总结，形成了遗传算法的基本框架。

13.1　遗传算法概要

13.1.1　遗传算法模型

对于一个求函数最大值的优化问题，一般可描述为下述数学规划模型：

$$\begin{aligned} \max \quad & f(\boldsymbol{X}) \\ \text{s.t.} \quad & \boldsymbol{X} \in \mathbf{R} \\ & \mathbf{R} \subseteq U \end{aligned}$$

式中，$\boldsymbol{X} = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$ 为决策变量；$f(\boldsymbol{X})$ 为目标函数；$\boldsymbol{U}$ 是基本空间，$\mathbf{R}$ 是 $\boldsymbol{U}$ 的一个子集。

遗传算法中，将 n 维决策向量 $\boldsymbol{X} = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$ 用 n 个记号 $X_i(i = 1, 2, \cdots, n)$ 所组成的符号串 $\boldsymbol{X}$ 来表示：

$$X = X_1 X_2 \cdots X_n \Rightarrow \boldsymbol{X} = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$$

把每一个 X_i 看作一个遗传基因，它的所有可能取值称为等位基因，这样，$\boldsymbol{X}$ 就可看作是由 n 个遗传基因所组成的一个染色体。染色体的长度可以是固定的，也可以是变化的。等位基因可以是一组整数，也可以是某一范围内的实数值，或者是记号。最简单的等位基因是由0和1这两个整数组成的，相应的染色体就可表示为一个二进制符号串。这种编码所形成的排列形式 $\boldsymbol{X}$ 是个体的基因型，与它对应的 $\boldsymbol{X}$ 值是个体的表现型。染色体 $\boldsymbol{X}$ 也称为个体 $\boldsymbol{X}$，对于每一个个体 $\boldsymbol{X}$，要按照一定的规则确定出其适应度。个体的适应度与其对应的个体表现型 $\boldsymbol{X}$ 的目标函数值相关联，$\boldsymbol{X}$ 越接近于目标函数的最优点，其适应度越大；反之，其适应度越小。

在遗传算法中，决策变量 X 组成了问题的解空间。对问题最优解的搜索是通过对染色体 X 的搜索过程来进行的，从而由所有的染色体 X 就组成了问题的搜索空间。

生物的进化是以集团为主体的。与此相对应，遗传算法的运算对象是由 M 个个体所组成的集合，称为群体。与生物一代一代的自然进化过程相似，遗传算法的运算过程也是一个反复迭代的过程，第 t 代群体记做 $P(t)$，经过一代遗传和进化后，得到第 $t+1$ 代群体，它们也是由多个个体组成的集合，记做 $P(t+1)$。这个群体不断地经过遗传和进化操作，并且每次都按照优胜劣汰的规则将适应度较高的个体更多地遗传到下一代，这样最终在群体中将会得到一个优良的个体 X，它所对应的表现型 X 将达到或接近于问题的最优解。

生物的进化过程主要是通过染色体之间的交叉和染色体的变异来完成的。遗传算法中最优解的搜索过程也模仿生物的这个进化过程，使用所谓的遗传算子（Genetic Operators）作用于群体 $P(t)$ 中，进行下述遗传操作，从而得到新一代群体 $P(t+1)$。

(1) 选择（Selection）：根据各个个体的适应度，按照一定的规则或方法，从第 t 代群体 $P(t)$ 中选择出一些优良的个体遗传到下一代群体 $P(t+1)$ 中。

(2) 交叉（Crossover）：将群体 $P(t)$ 内的各个个体随机搭配成对，对每一个个体，以某个概率（称为交叉概率，Crossover Rate）交换它们之间的部分染色体。

(3) 变异（Mutation）：对群体 $P(t)$ 中的每一个个体，以某一概率（称为变异概率，Mutation Rate）改变某一个或一些基因座上基因值为其他的等位基因。

13.1.2 遗传算法的特点

遗传算法是一类可用于复杂系统优化计算的鲁棒搜索算法，与其他一些优化算法相比，主要有下述几个特点：

(1) 遗传算法以决策变量的编码作为运算对象。传统的优化算法往往直接利用决策变量的实际值本身进行优化计算；但遗传算法不是直接以决策变量的值，而是以决策变量的某种形式的编码为运算对象，从而可以很方便地引入和应用遗传操作算子。

(2) 遗传算法直接以目标函数值作为搜索信息。传统的优化算法往往不只需要目标函数值，还需要目标函数的导数等其他信息。这样对许多目标函数无法求导或很难求导的函数，遗传算法就比较方便。

(3) 遗传算法同时进行解空间的多点搜索。传统的优化算法往往从解空间的一个初始点开始搜索，这样容易陷入局部极值点。遗传算法进行群体搜索，而

且在搜索的过程中引入遗传运算，使群体可以不断进化。这些是遗传算法所特有的一种隐含并行性。

（4）遗传算法使用概率搜索技术。遗传算法属于一种自适应概率搜索技术，其选择、交叉、变异等运算都是以一种概率的方式来进行的，从而增加了其搜索过程的灵活性。实践和理论都已证明了在一定条件下遗传算法总是以概率 1 收敛于问题的最优解。

13.1.3 遗传算法的发展

20 世纪 60 年代，美国密植安大学的 Holland 教授及其学生们受到生物模拟技术的启发，创造出了一种基于生物遗传和进化机制的适合于复杂系统计算优化的自适应概率优化技术——遗传算法。下面是在遗传算法的发展进程中一些关键人物所作出的一些主要贡献。

1. J. H. Holland

20 世纪 60 年代，Holland 认识到了生物的遗传和自然进化现象与人工自适应系统的相似关系，运用生物遗传和进化的思想来研究自然与人工自适应系统的生成以及它们与环境的关系，提出在研究和设计人工自适应系统时，可以借鉴生物遗传的机制，以群体的方法进行自适应搜索，并且充分认识到了交叉、变异等运算策略在自适应系统中的重要性。20 世纪 70 年代，Holland 提出了遗传算法的基本定理——模式定理（Schema Theorem），奠定了遗传算法的理论基础。1975 年，Holland 出版了第一本系统论述遗传算法和人工自适应系统的专著《自然系统和人工系统的自适应性》(Adaptation in Natural and Artificial Systems)。20 世纪 80 年代，Holland 实现了第一个基于遗传算法的机器学习系统——分类器系统，开创了基于遗传算法学习的新概念，为分类器系统构造出了一个完整的框架。

2. J. D. Bagley

1967 年，Holland 的学生 Bagley 在其博士论文中首次提出了“遗传算法”一词，并发表了遗传算法应用方面的第一篇论文。他发展了复制、交叉、变异、显性、倒位等遗传算子，在个体编码上使用了双倍体的编码方法。这些都与目前遗传算法中所使用的算子和方法类似。他还敏锐地意识到了在遗传算法执行的不同阶段可以使用不同的选择率，这将有利于防止遗传算法的早熟现象，从而创立了自适应遗传算法的概念。

3. K. A. De Jong

1975 年，De Jong 在其博士论文中结合模式定理进行了大量的纯数值函数优化计算实验，树立了遗传算法的工作框架，得到了一些重要且具有指导意义的结

论。他推荐了在大多数优化问题中都比较适用的遗传算法参数，还建立了著名的 De Jong 五函数测试平台，定义了评价遗传算法性能的在线指标和离线指标。

4. D. J. Goldberg

1989 年，Goldberg 出版了专著《搜索、优化和机器学习中的遗传算法》。该书系统总结了遗传算法的主要研究成果，全面而完整地论述了遗传算法的基本原理及其应用。

5. L. Davis

1991 年，Davis 编辑出版了《遗传算法手册》，书中包含了遗传算法在科学计算、工程技术和社会经济中的大量应用实例，该书为推广和普及遗传算法的应用起到了重要的指导作用。

6. J. R. Koza

1992 年，Koza 将遗传算法应用于计算机程序的优化设计及自动生成，提出了遗传编程的概念。Koza 成功地将提出的遗传编程方法应用于人工智能、机器学习、符号处理等方面。

13.1.4 遗传算法的应用

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的鲁棒性，所以广泛应用于很多学科。下面列举一些遗传算法的主要应用领域。

（1）函数优化。函数优化是遗传算法的经典应用领域，也是对遗传算法进行性能测试评价的常用算例。对于一些非线性、多模型、多目标的函数优化问题，用其他优化方法较难求解，而用遗传算法却可以方便地得到较好的结果。

（2）组合优化。遗传算法是寻求组合优化问题满意解的最佳工具之一。实践证明，遗传算法对于组合优化问题中的 NP 完全问题非常有效。

（3）生产调度问题。生产调度问题在很多情况下所建立起来的数学模型难以精确求解，即使经过一些简化之后可以进行求解也会因简化得太多而使求解结果与实际相差太远。现在遗传算法已经成为解决复杂调度问题的有效工具。

（4）自动控制。遗传算法已经在自动控制领域中得到了很好的应用，例如基于遗传算法的模糊控制器的优化设计，基于遗传算法的参数辨识，基于遗传算法的模糊控制规则的学习，利用遗传算法进行人工神经网络的结构优化设计和权值学习等。

（5）机器人学。机器人是一类复杂的难以精确建模的人工系统，而遗传算法的起源就来自于对人工自适应系统的研究，所以机器人学自然成为遗传算法的一个重要应用领域。

(6) 图像处理。图像处理是计算机视觉中的一个重要研究领域。在图像处理过程中，如扫描、特征提取、图像分割等不可避免地存在一些误差，这些误差会影响图像处理的效果。如何使这些误差最小是使计算机视觉达到实用化的重要要求，遗传算法在这些图像处理的优化计算方面得到了很好的应用。

(7) 人工生命。人工生命是用计算机、机械等人工媒体模拟或构造出的具有自然生物系统特有行为的人造系统。自组织能力和自学习能力是人工生命的两大重要特征。人工生命与遗传算法有着密切的关系，基于遗传算法的进化模型是研究人工生命现象的重要理论基础。

(8) 遗传编程。Koza 发展了遗传编程的概念，他使用了以 LISP 语言所表示的编码方法，基于对一种树形结构所进行的遗传操作来自动生成计算机程序。

(9) 机器学习。基于遗传算法的机器学习，在很多领域中都得到了应用。例如基于遗传算法的机器学习可用来调整人工神经网络的连接权，也可以用于人工神经网络的网络结构优化设计。

13.1.5 基本遗传算法

基本遗传算法（Simple Genetic Algorithms，SGA）是一种统一的最基本的遗传算法，它只使用选择、交叉、变异这三种基本遗传算子，其遗传进化操作过程简单，容易理解，是其他一些遗传算法的雏形和基础，它不仅给各种遗传算法提供了一个基本框架，同时也具有一定的应用价值。

1. 本遗传算法的构成要素

(1) 染色体编码方法。基本遗传算法使用固定长度的二进制符号串来表示群体中的个体，其等位基因是由二值符号集{0,1}组成的。初始群体中各个个体的基因值可用均匀分布的随机数来生成。

(2) 个体适应度评价。基本遗传算法按与个体适应度成正比的概率来决定当前群体中每个个体遗传到下一代群体中的机会的多少。为正确计算这个概率，这里要求所有个体的适应度必须为正数或零。

(3) 遗传算子。基本遗传算法使用下述三种遗传算子：选择运算使用比例选择算子，交叉运算使用单点交叉算子，变异运算使用基本位变异算子或均匀变异算子。

(4) 基本遗传算法的运行参数。基本遗传算法有下述 4 个运行参数需要提前设定：群体大小 M，即群体中所含个体数目，一般取为 20 ~ 100；遗传运算的终止进化代数 T，一般取为 100 ~ 500；交叉概率 P_c，一般取为 0.4 ~ 0.99；变异概率 P_m，一般取为 0.0001 ~ 0.1。

(5) 基本遗传算法的形式化定义。基本遗传算法可定义为一个 8 元组：

$$SGA=(C,E,P_0,M,\Phi,\Gamma,\Psi,T)$$

式中，C 为个体的编码方法；E 为个体适应度评价函数；P_0 为初始群体；M 为群体大小；Φ 为选择算子；Γ 为交叉算子；Ψ 为变异算子；T 为遗传运算终止条件。

2. 基本遗传算法的实现

（1）个体适应度评价。在遗传算法中，以个体适应度的大小来确定该个体被遗传到下一代群体中的概率。个体适应度越大，该个体被遗传到下一代的概率也越大；反之，个体的适应度越小，该个体被遗传到下一代的概率也越小。基本遗传算法使用比例选择算子来确定群体中各个个体遗传到下一代群体中的数量。为正确计算不同情况下各个个体的遗传概率，要求所有个体的适应度必须为正数或零，不能是负数。

为满足适应度取非负值的要求，基本遗传算法一般采用下面两种方法之一将目标函数值 $f(x)$ 变换为个体的适应度 $F(x)$。

方法 1：对于目标函数是求极大化，方法为：

$$F(X)=\begin{cases}f(X)+C_{\min}, & \text{当} f(X)+C_{\min}>0 \text{ 时}\\ 0, & \text{当} f(X)+C_{\min}\leqslant 0 \text{ 时}\end{cases}$$

式中，$C_{\min}$ 为一个适当地相对比较小的数，它可用下面几种方法之一来选取：预先指定的一个较小的数；进化到当前代为止的最小目标函数值；当前代或最近几代群体中的最小目标值。

方法 2：对于求目标函数最小值的优化问题，变换方法为：

$$F(X)=\begin{cases}C_{\max}-f(X), & \text{当} f(X)<C_{\max} \text{ 时}\\ 0, & \text{当} f(X)\geqslant C_{\max} \text{ 时}\end{cases}$$

式中，$C_{\max}$ 为一个适当地相对比较大的数，它可用下面几种方法之一来选取：预先指定的一个较大的数；进化到当前代为止的最大目标函数值；当前代或最近几代群体中的最大目标值。

（2）比例选择算子。比例选择实际上是一种有退还的随机选择，也叫做赌盘（Roulette Wheel）选择，因为这种选择方式与赌博中的赌盘操作原理非常相似。

比例选择算子的具体执行过程是：先计算出群体中所有个体的适应度之和；其次计算出每个个体的相对适应度的大小，此值即为各个个体被遗传到下一代群体中的概率；最后再使用模拟赌盘操作（即 0 到 1 之间的随机数）来确定各个个体被选中的次数。

（3）单点交叉算子。单点交叉算子是最常用和最基本的交叉操作算子。单点交叉算子的具体执行过程如下：对群体中的个体进行两两随机配对；对每一对

相互配对的个体，随机设置某一基因座之后的位置为交叉点；对每一对相互配对的个体，依设定的交叉概率 P_c 在其交叉点处相互交换两个个体的部分染色体，从而产生出两个新个体。

（4）基本位变异算子。基本位变异算子的具体执行过程为：对个体的每一个基因座，依变异概率 P_m 指定其为变异点；对每一个指定的变异点，对其基因值做取反运算或用其他等位基因值来代替，从而产生出一个新的个体。

3. 遗传算法的应用步骤

遗传算法提供了一种求解复杂系统优化问题的通用框架。对于具体问题，可按下述步骤来构造：

（1）确定决策变量及其各种约束条件，即确定出个体的表现型 X 和问题的解空间。

（2）建立优化模型，即描述出目标函数的类型及其数学描述形式或量化方法。

（3）确定表示可行解的染色体编码方法，即确定出个体的基因型 X 及遗传算法的搜索空间。

（4）确定解码方法，即确定出由个体基因型 X 到个体表现型 X 的对应关系或转换方法。

（5）确定个体适应度的量化评价方法，即确定出由目标函数值 $f(x)$ 到个体适应度 $F(x)$ 的转换规则。

（6）设计遗传算子，即确定出选择运算、交叉运算、变异运算等遗传算子的具体操作方法。

（7）确定遗传算法的有关运行参数，即确定出遗传算法的 M，T，P_c，P_m 等参数。

13.1.6　遗传算法的模式定理

Holland 提出的模式定理（Schema Theorem），是遗传算法的基本原理，从进化动力学的角度提供了能够较好地解释遗传算法机理的一种数学工具，同时也是编码策略、遗传策略等分析的基础。

1. 模式与模式空间

遗传算法将实际问题表示成位串空间，以群体为基础，根据适者生存的原则，从中选择出高适应值的位串进行遗传操作，产生出下一代适应性好的位串集合，从而将整个群体不断转移到位串空间中适应值高的子集上，直到获得最优解。在这一过程中，群体中是由哪些信息来指导和记忆寻优过程的呢？Holland 发现，位串中的某些等位基因的连接与适应值函数之间存在着某种联系，这种联

系提供了寻优过程的指导信息，引导着群体在位串空间中的移动方向。

遗传算法在工作过程中，建立并管理着问题参数空间、位串空间（或者称为编码空间）、模式空间和适应值空间等 4 个空间及其之间的转换关系，如图 13-1 所示。

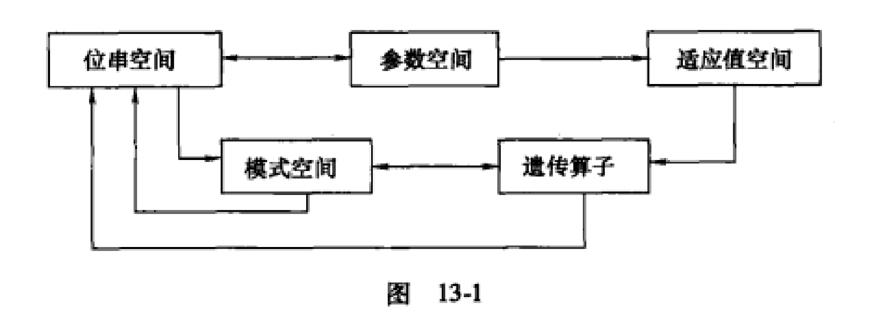


图 13-1

（1）模式空间。采用字符集 $K=\{0,1\}$ 对位串参数进行二进制编码，位串空间表示为 $S^L=\{0,1\}^L$，该空间的基数为 $|S^L|=2^L$。

扩展字符集 $K'=\{0,1,*\}$，其中 * 是通配符，即可和 0 或 1 匹配。扩展位串空间表示为 $S_e^L=\{0,1,*\}^L$，该空间的基数为 $|S_e^L|=3^L$。则称 S_e^L 为 S^L 的模式空间。显然，包含 2^L 个位串的位串空间，对应着 3^L 个模式位串的模式空间。

（2）模式。扩展位串空间 $S_e^L=\{0,1,*\}^L$ 中的任何一个点，称为对应于位串空间 $S^L=\{0,1\}^L$ 的一个模式（Schema）。

模式是由 S^L 中具有共同特征的位串所组成的集合，它描述了该集合中位串上的共同基因特征。

例如，模式 00 * * 表示位串程度为 4，两个高位基因为 00 的位串集合，即 $\{0000,0001,0010,0011\}$。

（3）模式的阶。模式的阶（Schema Order）是指模式中所含有 0、1 确定基因位的个数，记作 $O(H)$。

（4）模式的定义长度。模式的定义长度（Defining Length）是指模式中从左到右第一个非 * 位和最后一个非 * 位之间的距离，记作 $\delta(H)$。

（5）模式的维数。模式的维数（Schema Dimension）是指模式中所包含的位串的个数，也称为模式的容量，记作 $D(H)$，$D(H)=2^{L-O(H)}$。

（6）模式的适应值。模式在 t 代群体中包含的个体位串为 $\{a_1,a_2,\cdots,a_m\}$，其中 $m=m(H,t)$ 为模式 H 在第 t 代群体中所包含位串数量，称为模式 H 在群体中的生存数量（Survials）或者采样样本（Samples），$a_j\in H(j=1,2,\cdots,m)$，则模式 H 在第 t 代群体中的适应值估计（简称模式的适应值）为：

$$f(H,t) = \sum_{j=1}^{m} f(a_j)/m$$

从编码空间来看，$m(H,t)$是当前群体中包含于模式 H 的个体数量，反映了所对应的模式空间的分布情况。该数量越大，说明群体越集中于模式 H 代表的子空间。从模式空间来看，$m(H,t)$是模式 H 在当前群体中的个体采样数量，反映了所对应的编码空间的分布情况。该数量越大，说明群体中的个体越趋向相似和一致，在编码空间的搜索范围越小。

例如，模式 H = * 101 *，则 $O(H)=3$，$\delta(H)=2$，$D(H)=2^{L-O(H)}=2^{5-3}=2^2=4$。可见，一个模式 H 由位串长度 L、阶 $O(H)$、定义长度 $\delta(H)$、容量 $D(H)$ 和适应值 $f(H,t)$ 等 5 个指标来描述。

2. 模式生存模型

遗传算法在群体进化过程中，可以看作是通过选择、交叉和变异算子，不断发现重要基因、寻找较好模式的过程。高适应值的个体被选择概率大于低适应值的个体。同样，根据模式适应值的定义，选择算子对于模式的作用表现为：其适应值越高，被选择的概率也就越大，所以好的模式在群体中的个体采样数量会不断增加，其上的重要基因或者有效基因也得以遗传下来；对交叉算子来讲，如果它不分割一个模式，则该模式不变，反之可以导致模式消失或所包含的高适应值个体数量减少，同时交叉算子还可以创建新的模式。

变异算子的变异率很小，对模式生成和破坏的概率也很小。

假设 P_t 为第 t 代规模为 n 的群体，则 $P(t)=\{a_1(t),a_2(t),\cdots,a_n(t)\}$。

(1) 选择算子对模式 H 生存数量的影响。假定在 t 代群体中模式 H 的生存数量为 $m(H,t)$，在选择操作过程中，个体按概率

$$p_i = \frac{f(a_i)}{\sum_{i=1}^{n} f(a_i)}$$

被选择，则在第 $t+1$ 代，模式 H 的生存数量为：

$$m(H,t+1) = \frac{m(H,t) \times n \times f(H)}{\sum_{i=1}^{n} f(a_i)}$$

将群体的平均适应值表示为：

$$\bar{f} = \frac{\sum_{i=1}^{n} f(a_i)}{n}$$

故上式可表示为：

$$m(H,t+1)=m(H,t)\times\frac{f(H)}{\bar{f}}$$

该式说明下一代群体中模式 H 的生存数量与模式的适应值成正比，与群体平均适应值成反比。当 $f(H)>\bar{f}$ 时，H 的生存数量增加；当 $f(H)<\bar{f}$ 时，H 的生存数量减少。群体中任一模式的生存数量都将在选择操作中按上式规律变化。

设 $f(h)-\bar{f}=c\bar{f}$，其中 c 为常数，则公式变为：

$$m(H,t+1)=m(H,t)\times\frac{(\bar{f}+c\bar{f})}{\bar{f}}=m(H,t)\times(1+c)$$

群体从 $t=0$ 开始选择操作，假设 c 保持固定不变，则上式可以表示为：

$$m(H,t)=m(H,0)\times(1+c)^t$$

可以看出，在选择算子作用下，模式的生存数量是以迭代次数为指数函数方式进行变化的。当 $c>0$ 时，模式的生存数量以指数规律增加；当 $c<0$ 时，生存数量以指数规律减少。这种变化仅仅是已有模式生存数量的变化而已，并没有产生新的模式。

（2）交叉算子对模式 H 生存数量的影响。交叉操作对模式的影响与其定义长度 $\delta(H)$ 有关。$\delta(H)$ 越大，模式被破坏的可能性越大。

若染色体位串长度为 L，在单点交叉算子作用下模式 H 的存活概率为：

$$p_{\text{survial}}=1-\frac{\delta(H)}{L-1}$$

在交叉概率为 p_c 的单点交叉算子作用下，该模式的存活概率为：

$$p_{\text{survial}}\geqslant 1-p_c\times\frac{\delta(H)}{L-1}$$

那么，模式 H 在选择、交叉算子共同作用下的生存数量可用下式计算：

$$m(H,t+1)\geqslant m(H,t)\times\frac{f(H)}{\bar{f}}\times p_{\text{survial}}=m(H,t)\times\frac{f(H)}{\bar{f}}\times\frac{1-p_c\times\delta(H)}{L-1}$$

可见，在选择算子、交叉算子共同作用下，模式生存数量的变化与其平均适应值及定义长度 $\delta(H)$ 密切相关。当 $f(H)>\bar{f}$，且 $\delta(H)$ 较小时，群体中该模式生存数量以指数规律增长；反之则以指数规律减少。

（3）变异算子对模式 H 生存数量的影响。对于群体中的任一个体，变异操作就是以概率 P_m 随机改变某一基因位的等位基因。为了使模式 H 在变异操作中生存下来，其上所有确定位的等位基因均不发生变化的概率为 $(1-P_m)^{O(H)}$。一般情况下 $P_m\leqslant 1$，所以模式 H 的生存概率可近似表示为：

$$(1-P_m)^{O(H)}=1-p_m\times O(H)$$

那么，在选择、变异算子的共同作用下，模式的生存数量为：

$$m(H,t+1)=m(H,t)\times\frac{f(H)}{\bar{f}}\times(1-p_m\times O(H))$$

综合考虑选择、交叉和变异算子的共同作用，模式的生存数量可表示为：

$$m(H,t+1)\geqslant m(H,t)\times\left(\frac{f(H)}{\bar{f}}\times(1-p_c\times\delta(H)/(L-1))\times(1-p_m\times O(H))\right)$$

忽略高次极小项：

$$((p_c\times\delta(H)/(L-1))\times(p_m\times O(H)))$$

上式变为：

$$m(H,t+1)\geqslant m(H,t)\times\frac{f(H)}{\bar{f}}\times(1-p_c\times\delta(H)/(L-1)-p_m\times O(H))$$

3. 模式定理

通过以上关于三个遗传算子对生存模式数量的影响分析，可以得出如下“模式定理”：在选择、交叉、变异算子的作用下，那些低阶、定义长度短、超过群体平均适应值的模式的生存数量，将随着迭代次数的增加以指数规律增长。

这就是由 Holland 提出的模式定理，称之为遗传算法进化动力学的基本原理。该定理反映了重要基因的发现过程。重要基因的缔结对应于较高的适应值，说明了它们所代表的个体在下一代有较高的生存能力，是提高群体适应性的进化方向。

13.2　Genetic Algorithm Toolbox

13.2.1　函数概述

遗传算法通过不同的编码方式、不同的进化策略可以求解多类型的优化问题，MATLAB 专门提供了遗传算法与直接搜索工具箱（Genetic Algorithm and Direct Search Toolbox），为了用户使用方便，还设计了用户界面（GUI），如图 13-2 所示。本章只对其中的 Genetic Algorithm Toolbox 进行介绍。

Genetic Algorithm Toolbox 计算的标准形式为：

$$\min_{x} f(\boldsymbol{x})$$

$$\text{s.t.}\quad \begin{aligned} &\boldsymbol{c}(\boldsymbol{x})\leqslant 0\\ &\boldsymbol{q}(\boldsymbol{x})=0\\ &\boldsymbol{A}\cdot\boldsymbol{x}\leqslant\boldsymbol{b}\\ &\boldsymbol{Aeq}\,\boldsymbol{x}=\boldsymbol{beq}\\ &\boldsymbol{lb}\leqslant\boldsymbol{x}\leqslant\boldsymbol{ub}\end{aligned}$$

图 13-2

这里 $\boldsymbol{x}, \boldsymbol{b}, \boldsymbol{beq}, \boldsymbol{lb}, \boldsymbol{ub}$ 为向量，$\boldsymbol{A}$ 为 $\boldsymbol{Aeq}$ 为矩阵，$f(\boldsymbol{x})$ 为目标函数，$\boldsymbol{c}(\boldsymbol{x}), \boldsymbol{ceq}(\boldsymbol{x})$ 为非线性约束，$\boldsymbol{Ax} \leqslant \boldsymbol{b}$，$\boldsymbol{Aeq}\ \boldsymbol{x} = \boldsymbol{beq}$ 为线性约束，$\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$ 为可行解的区间约束。Genetic Algorithm 是启发式算法，算法对目标函数与约束函数的函数性质没特别要求。

13.2.2 函数使用说明及示例

下面将用具体示例具体说明函数的使用方法。例如优化问题：

$$\min f = (1 - x_1^2 + x_1^{4/3}) x_1^2 + x_1 x_2 + (x_1^2 - 1) x_2^2$$

$$\text{s.t.} \quad x_1^2 + x_2^2 \leqslant 1$$

$$x_1 + x_2 = 2$$

（1）编写目标函数 GAobjfun1.m

```
functionf = GAobjfun1(x)
f = (1 - x(1)^2 + x(1)^4/3) * x(1)^2 + x(1) * x(2) + ( -1 + x(2)^2) * x(2)
^2;
```

（2）约束函数 GaConfun.m

```
function[c,ceq] = GaConfun(x);
c = x(1)^2 + x(2)^2 - 1;(不等式约束)
ceq = x(1) + x(1) - 2;(等式约束)
```

（3）在 MATLAB 的命令窗口（Command Window）输入 gatool 调用 Genetic Algorithm Toolbox（见图 13-3）。

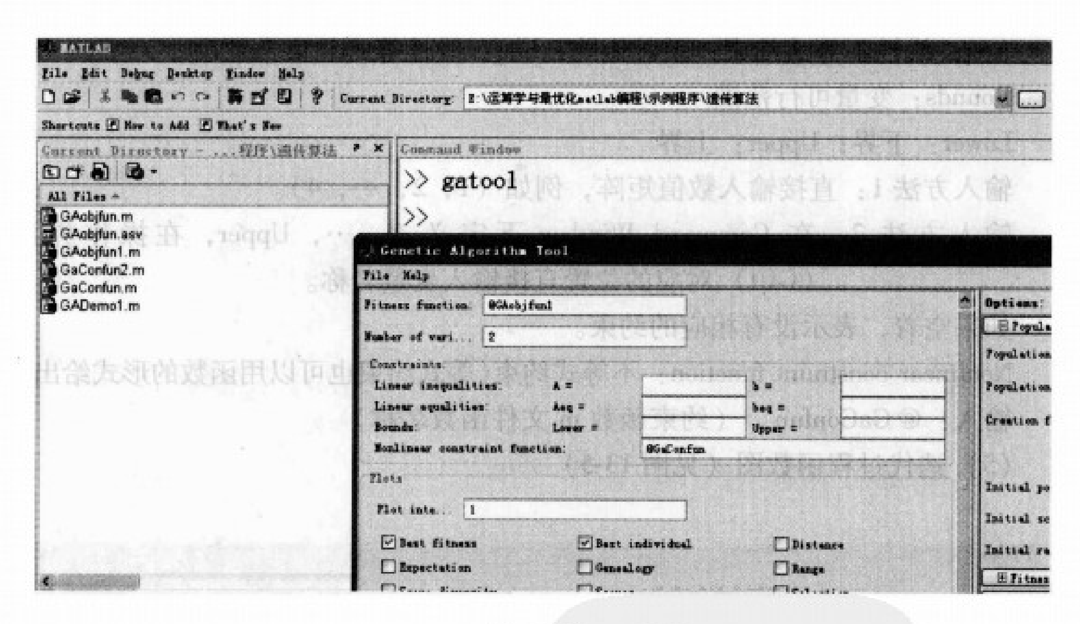


图 13-3

（4）输入目标函数与约束条件（见图 13-4）

Genetic Algorithm Tool
File Help
Fitness function: @GAobjfun1
Number of vari... 2
Constraints:
Linear inequalities: A = b =
Linear equalities: Aeq = beq =
Bounds: Lower = Upper =
Nonlinear constraint function: @GaConfun

图 13-4

Fintness function：适应函数，即目标函数

输入：@ GAobjfun1　　　（目标函数的 m 文件函数名）

Numer of vari….：变量个数，即优化问题中变量个数

输入：2　　　　　　　　（示例问题的变量个数为 2）

Linear inequalities：线性不等式约束

A：线性不等式约束，系数矩阵；b：线性不等式约束常数项

Linear equalities：线性等式约束

Aeq：线性不等式约束，系数矩阵；beq：线性等式约束常数项

Bounds：变量可行范围

Lower：下界；Upper：上界

输入方法 1：直接输入数值矩阵，例如（1，2，…，4）。

输入方法 2：在 Command Window 下定义 A，…，Upper，在操作界面（GUI）对应的位置直接输入变量名称。

如果空着，表示没有相应的约束。

Nonlinear constraint function：不等式约束（等式约束也可以用函数的形式给出）

输入：@ GaConfun　（约束函数 m 文件函数名称）

（5）迭代过程函数图（见图 13-5）

图　13-5

Plot inte…图像更新频率

输入：1，表示算法迭代一次图像更新一次

Best fitness：最优结果图像

Best individual：最优个体

Distance：种群个体间距离

Expectation：个体进化比率

Genealogy：个体宗系图

Range：适应函数值

Score Diversity：种群得分柱状图

Scores：每代种群适应性得分

Selection：显示哪些个体被选择

Stopping：停止条件准则等级

Max constraint：显示符合约束的程度

Custom function：自定义图像

具体可以参考：Genetic Algorithm Tool Quick reference（见图 13-6）

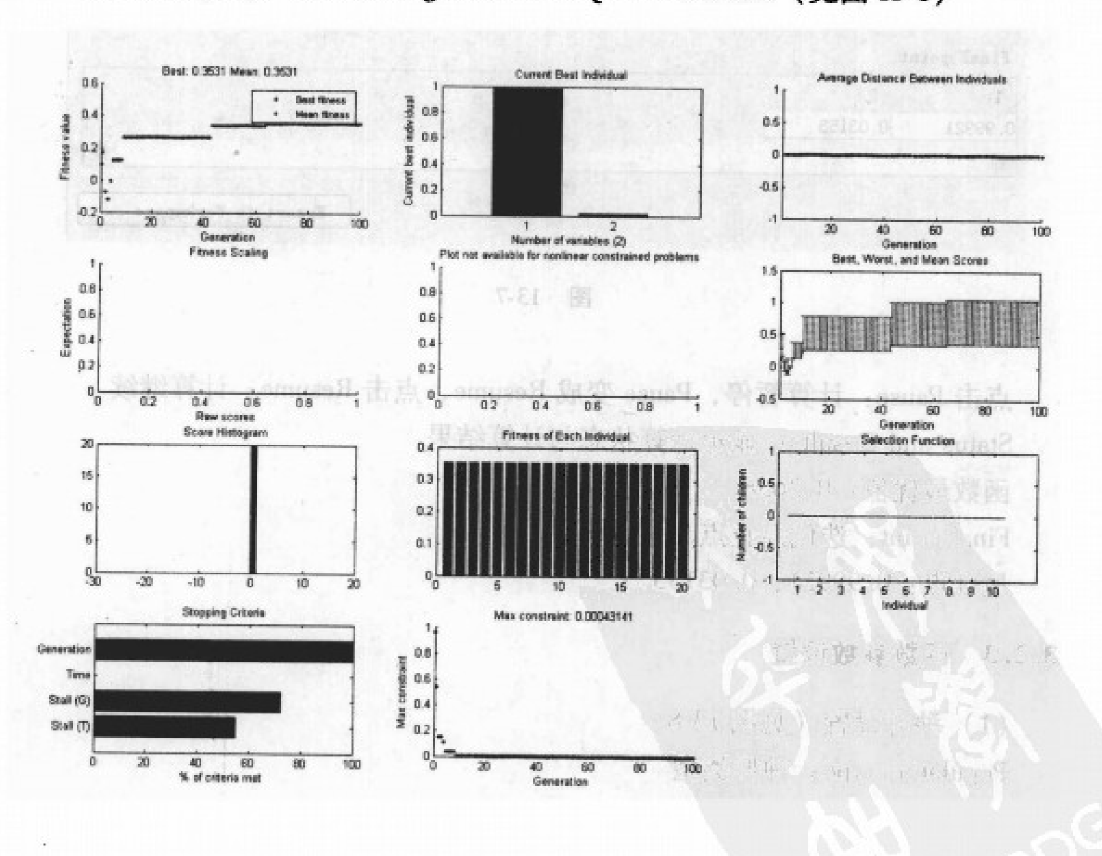


图 13-6

（6）算法迭代过程控制及计算结果（见图 13-7）

Use random states from previousrun：使用先前迭代过程的随机状态

点击 Start：开始计算

图 13-7

点击 Pause：计算暂停，Pause 变成 Resume，点击 Resume：计算继续
Status and Results：显示计算状态与计算结果
函数最优值：0.3638
Final point：迭代停止点
最优点：0.99921，0.03153

13.2.3 函数参数设置

（1）种群设置（见图 13-8）
Population type：种群类型
Double Vector：实数向量性
Bit String：0－1 串型
Custom：自定义型
Population size：种群规模
Creation Funct…：种群个体生成方法
Uniform：均匀分布生成法
Custom：自定义型

Initial popula…：初始种群

Initial scores：种群初始适应度评分

Initial range：初始适应函数值

图　13-8

（2）适应度评分方法（见图 13-9）

图　13-9

Scaling function：评分函数

Rank：排序法评分

Proportional：进化度评分法（子个体比其母个体进化的程度）

……

（3）选择策略（见图 13-10）

图 13-10

Stochastic uniform：均匀随机法
Roulette：轮盘赌法
Tournament：竞争选择法
Custom：自定义选择法

（4）繁殖策略（见图 13-11）

图 13-11

Elite count：产生新个体的母个体数
Crossover fract…：交叉概率

（5）停止条件（见图 13-12）

Generations：迭代次数
Time limit：时间限制
Fitness limit：适应度限制
Stall generations：最优个体不再进化次数限制（如果在××次迭代中最优个体不再进化则迭代停止。
Stall time limit：最优个体不再进化时间限制
Function tolerance：目标函数误差控制范围
Nonlinear constrain tolera…：非线性约束函数误差控制范围

⊟ Stopping criteria

Generations:	100
Time limit:	Inf
Fitness limit:	-Inf
Stall generations:	50
Stall time limit:	20
Function tolerance:	1e-006
Nonlinear constraint tolera...	1e-006

图 13-12

(6) 其他参数设置（见图13-13）

Options:

⊟ Population

Population type:	Bit string
Population size:	20
Creation funct...:	Uniform
Initial popula...:	[]
Initial scores:	[]
Initial range:	[0 ; 1]

⊟ Fitness scaling

Scaling function:	Rank

⊞ Selection

⊞ Reproduction

⊞ Mutation

⊞ Crossover

⊞ Migration

⊞ Algorithm settings

⊞ Hybrid function

⊞ Stopping criteria

Quick reference:

performs.

Time limit specifies the maximum time in seconds the genetic algori before stopping.

Fitness limit - If the best fitness value is less than or equal to the val **Fitness limit**, the algorithm stops.

Stall generations - If the weighted average change in the fitness fun value over Stall Generations is less than Function Tolerance, the algor stops.

Stall time limit - If there is no improvement in the best fitness value interval of time in seconds specified by **Stall time limit**, the algorithm

Function tolerance - If the cumulative change in the fitness function over Stall Generations is less than Function Tolerance, the algorithm s

Nonlinear constraint tolerance specifies the termination tolerance f maximum nonlinear constraint violation.

Output Function Options

History to new window outputs the iterative history of the algorithm separate window.

Interval specifies the number of generations between successive out

Custom enables you to write you own output function.

Display to Command Window Options

Level of display specifies the amount of information displayed in the Command Window when you run the genetic algorithm. You can cho the following options:

- Off - No output is displayed.
- Iterative - Information is displayed at each iteration of the algorithm.
- Diagnose - Information is displayed at each iteration. In addition, the diagnostic lists some problem information and the options that are changed from the defaults.
- Final - The reason for stopping is displayed.

图 13-13

其他参数设置，如变异策略、算法设置与混合函数设置具体可以参考 Quick reference。

13.2.4 遗传算法 M 文件自动生成

在 Genetic Algorithm tool→File→Generate M-file 可以生成该求解问题的 M 文件，比如命名为 GADemo1. m，存在指定工作目录下，以后可以随时调用。

GADemo1. m 程序

```
function[X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = GADemo1
%% This is an auto generated M file to do optimization with the Genetic Algo-
rithm and
% Direct Search Toolbox. Use GAOPTIMSET for default GA options structure.
%% Fitness function
fitnessFunction = @GAobjfun1;
%% Number of Variables
nvars = 2;
% Linear inequality constraints
Aineq = [];
Bineq = [];
% Linear equality constraints
Aeq = [];
Beq = [];
% Bounds
LB = [];
UB = [];
% Nonlinear constraints
nonlconFunction = @GaConfun;
% Start with default options
options = gaoptimset;
%% Modify some parameters
options = gaoptimset(options,'MutationFcn',{@mutationgaussian 1 1});
options = gaoptimset(options,'Display','off');
options = gaoptimset(options,'PlotFcns',{@gaplotbestf@gaplotbestindiv});
% Set the states of random number generators
rand('state',[0.29901;0.083894;0.99927;0.33314;0.53815;0.48539;0.88538;0.31101;
```

```
0.57405;0.2643;0.68191;0.98536;0.60708;0.69837;0.98953;0.32555;
0.92429;0.71165;0.38552;0.0097387;0.0075905;0.61386;0.77373;
0.7406;0.94044;0.55742;0.3987;0.78401;0.51979;0.92262;0.55007;
0.54655;1.1102e-016;3.9968e-015;3.2582e-007]);
randn('state',[2753740601;618774148]);
%% Run GA
[X,FVAL,REASON,OUTPUT,POPULATION,SCORES]
            ga(fitnessFunction,nvars,Aineq,Bineq,Aeq,Beq,LB,UB,nonl-
            conFunction,options);
```

附录　MATLAB 优化工具箱参数设置

1. 优化工具箱参数说明

优化工具箱参数设置说明格式

序号　参数名称

参数描述

计算规模

与该参数有关的 MATLAB 优化函数

(1) BranchStrategy　分支选择策略

Strategy bintprog uses to select branch variable　整数规划分支变量选择

M　中等规模计算

Bintprog　整数规划函数

(2) DerivativeCheck　导数检验

Compare user-supplied analytic derivatives(gradients or Jacobian) to finite differencing derivatives　对有限的导数(梯度或 Jacobian 矩阵)进行分析

B　中等规模或大规模计算

fgoalattain,fmincon,fminimax,fminunc,fseminf,fsolve,lsqcurvefit,lsqnonlin

无约束优化,约束优化等相关优化函数

(3) Diagnostics　诊断结果

Display diagnostic information about the function to be minimized or solved.

输出函数最小化或求解得诊断信息

B　中等规模或大规模计算

All but fminbnd,fminsearch,fzero,and lsqnonneg

除 fminbnd,fminsearch,fzero,lsqnonneg 的所有优化函数

(4) DiffMaxChange　导数最大变化

Maximum change in variables for finite differencing.

有限导数的变量最大变化,即如果导数超过该值程序报错

M　中等规模计算

fgoalattain,fmincon,fminimax,fminunc,fseminf,fsolve,lsqcurvefit,lsqnonlin

(5) DiffMinChange　导数最小变化

Minimum change in variables for finite differencing

有限导数的变量最小变化

M 中等规模计算

fgoalattain, fmincon, fminimax, fminunc, fseminf, fsolve, lsqcurvefit, lsqnonlin

(6) Display 计算过程输出

Level of display. 'off' displays no output; 'iter' displays output at each iteration; 'final' displays just the final output; 'notify' displays output only if function does not converge.

Off:关闭计算过程输出

Iter:输出每次迭代结果

Final:只输出最后结果

Notify:当算法不收敛的时候输出结果

B:中等规模或大规模计算

使用于 Opimization 的所有优化算法

(7) FunValCheck 函数值检验

Check whether objective function and constraints values are valid. 'on' displays a warning when the objective function or constraints return a value that is complex, NaN, or Inf. Note: FunValCheck does not return a warning for Inf when used with fminbnd, fminsearch, or fzero, which handle Inf appropriately.

检查函数值是否适合,如果是 NaN, Inf 则是不适合

B 中等规模或大规模计算

fgoalattain, fminbnd, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, fzero, lsqcurvefit, lsqnonlin

(8) GoalsExactAchieve

Specify the number of objectives required for the objective fun to equal the goal goal. Objectives should be partitioned into the first few elements of F.

M 中等规模计算

Fgoalattain

(9) GradConstr 约束函数梯度

User-defined gradients for the nonlinear constraints. 使用者给出的非线性约束函数梯度

M 中等规模计算

fgoalattain, fmincon, fminimax 目标优化,最大最小问题,约束最优化

(10) GradObj 目标函数梯度

User-defined gradients for the objective functions. 使用者给出的目标函数梯度

B 中等规模或大规模计算

fgoalattain, fmincon, fminimax, fminunc

(11) fseminfHessian Hessian 阵选取参数

If 'on', function uses user-defined Hessian or Hessian information (when using HessMult), for the objective function. If 'off', function approximates the Hessian using finite differences.

fseminfHessian 'no':使用用户定义的 Hessian 阵

L 大规模计算

Fmincon 约束优化函数

(12) fminuncHessMult 函数 fminunc 的 Hesse 乘子

User-defined Hessian multiply function. Avoiding Global Variables via Anonymous and Nested Functions explains how to parameterize the Hessian multiply function, if necessary.

L 大规模计算

fmincon, fminunc

(13) quadprogHessPattern 函数 quadprog 的 Hesse 伴随矩阵

Sparsity pattern of the Hessian for finite differencing. The size of the matrix is n-by-n, where n is the number of elements in x0, the starting point.

L 大规模计算

Fmincon

(14) fminuncHessUpdate 函数 fminunc 的 Hesse 阵的更新策略

Quasi-Newton updating scheme. 拟牛顿法的 Hesse 阵的更新策略,如 DFP、BFGS 等

M 中等规模计算

Fminunc 无约束优化

(15) InitialHessMatrix 初始 Hesse 阵

Initial quasi-Newton matrix. 拟牛顿法的初始 Hesse 阵

M 中等规模计算

Fminunc 无约束最优化

(16) InitialHessType 初始 Hesse 阵类型

Initial quasi-Newton matrix type. 拟牛顿法的初始 Hesse 阵类型

M 中等规模计算

Fminunc 无约束优化

(17) Jacobian　Jacobian 矩阵

If 'on', function uses user-defined Jacobian or Jacobian information (when using JacobMult), for the objective function. If 'off', function approximates the Jacobian using finite differences.

B　中等规模或大规模计算

fsolve, lsqcurvefit, lsqnonlin

(18) JacobMult Jacobian　矩阵乘子

User-defined Jacobian multiply function. Avoiding Global Variables via Anonymous and Nested Functions explains how to parameterize the Jacobian multiply function, if necessary.

L　大规模计算

fsolve, lsqcurvefit, lsqlin, lsqnonlin

(19) JacobPattern　Jacobian 伴随矩阵

Sparsity pattern of the Jacobian for finite differencing. The size of the matrix is m-by-n, where m is the number of values in the first argument returned by the user-specified function fun, and n is the number of elements in x0, the starting point.

L　大规模计算

fsolve, lsqcurvefit, lsqnonlin

(20) LargeScale　大规模模式

Use large-scale algorithm if possible. 必要的时候使用大规模算法

B　中等规模或大规模计算

fmincon, fminunc, fsolve, linprog, lsqcurvefit, lsqlin, lsqnonlin, quadprog

(21) LevenbergMarquardt LevenbergMarquardt 算法

Choose Levenberg-Marquardt over Gauss-Newton algorithm. 'on' specifies the Levenberg-Marquardt algorithm. 'off' specifies the Gauss-Newton algorithm.

'on':使用 LevenbergMarquardt 算法代替 Gauss-Newton 算法

'off':默认使用 Gauss-Newton 算法

M　中等规模计算

lsqcurvefit, lsqnonlin

(22) LineSearchType　线性搜索方式

Line search algorithm choice. 线性搜索方式选择

M　中等规模计算

fsolve, lsqcurvefit, lsqnonlin

(23) MaxFunEvals　最大目标函数计算次数

Maximum number of function evaluations allowed.

B　中等规模或大规模计算

fgoalattain, fminbnd, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, lsqcurvefit, lsqnonlin

(24) MaxIter　最大迭代次数

Maximum number of iterations allowed.

B　中等规模或大规模计算

All but fzero and lsqnonneg

(25) MaxNodes　最大节点数(整数规划)

Maximum number of possible solutions, or nodes, the binary integer programming function bintprog searches.

M　中等规模计算

Bintprog 整数规划

(26) MaxPCGIter　最大共轭梯度迭代

Maximum number of iterations of preconditioned conjugate gradients method allowed.

L　大规模计算

fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog

(27) MaxRLPIter　最大线性规划迭代次数(整数规划)

Maximum number of iterations of linear programming relaxation method allowed.

M　中等规模计算

Bintprog

(28) MaxSQPIter　最大序列二次归化次数

Maximum number of iterations of sequential quadratic programming method allowed.

M　中等规模计算

Fmincon

(29) MaxTime　函数最大计算时间

Maximum amount of time in seconds allowed for the algorithm.

M　中等规模计算

Bintprog　整数规划

(30) MeritFunction　价值函数

Use goal attainment/minimax merit function(multiobjective) vs. fmincon(single objective).

M 中等规模计算

fgoalattain, fminimax

(31) MinAbsMax

Number of F(x) to minimize the worst case absolute values.

M 中等规模计算

Fminimax

(32) NodeDisplayInterval 计算节点输出

Node display interval for bintprog.

M 中等规模计算

Bintprog 整数规划

(33) NodeSearchStrategy 节点搜索策略

Search strategy that bintprog uses.

M 中等规模计算

Bintprog 整数规划

(34) NonlEqnAlgorithm 非线性等式算法

Specify one of the following algorithms for solving nonlinear equations:

'dogleg'—Trust-region dogleg algorithm(default)默认信赖域法

'lm'—Levenberg-Marquardt Levenberg-Marquardt 法

'gn'—Gauss-Newton 高斯牛顿法

M 中等规模计算

Fsolve

(35) OutputFcn 输出函数

Specify a user-defined function that the optimization function calls at each iteration. See Output Function. 输出每次迭代目标函数值

B 中等规模或大规模计算

fgoalattain, fmincon, fminimax, fminunc, fseminf, fsolve, lsqcurvefit, lsqnonlin

(36) PrecondBandWidth

Upper bandwidth of preconditioner for PCG.

L 大规模计算

fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog

(37) RelLineSrchBnd 线性搜索步长

Relative bound on line search step length.

M 中等规模计算

fgoalattain, fmincon, fminimax, fseminf

(38) RelLineSrchBndDuration

Number of iterations for which the bound specified in RelLineSrchBnd should be active.

M　中等规模计算

fgoalattain, fmincon, fminimax, fseminf

(39) Simplex　单纯形法

If 'on', function uses the simplex algorithm. 是否使用单纯行法

M　中等规模计算

Linprog　线性规划

(40) TolCon　约束误差控制范围

Termination tolerance on the constraint violation.

B　中等规模或大规模计算

bintprog, fgoalattain, fmincon, fminimax, fseminf

(41) TolFun　目标函数误差控制范围

Termination tolerance on the function value.

B　中等规模或大规模计算

bintprog, fgoalattain, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, linprog(large-scale only), lsqcurvefit, lsqlin(large-scale only), lsqnonlin, quadprog(large-scale only)

(42) TolPCG　共轭梯度迭代的误差控制范围

Termination tolerance on the PCG iteration.

L　大规模计算

fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog

(43) TolRLPFun　线性松弛函数的误差控制范围

Termination tolerance on the function value of a linear programming relaxation problem.

M　中等规模计算

Bintprog

(44) TolX　变量 X 的误差控制范围

Termination tolerance on x.

B　中等规模或大规模计算

All functions except the medium-scale algorithms for linprog, lsqlin, and quadprog

(45) TolXInteger　整数变量 X 的误差控制范围

Tolerance within which bintprog considers the value of a variable to be an integer.

M 中等规模计算

Bintprog

(46) TypicalX 变量 X 的类型

Typical x values. The length of the vector is equal to the number of elements in x0, the starting point. 与初始可行点 x0 的类型一致

B 中等规模或大规模计算

fgoalattain, fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog

2. 优化工具箱参数设置方法

设置函数:

options = optimset('param1', value1, 'param2', value2, ...)

设置参数格式:

(1) Optimization options used by both large-scale and medium-scale algorithms

大规模或中等规模计算使用的参数设置

1) DerivativeCheck 'on' | {'off'}

2) Diagnostics 'on' | {'off'} Display 'off' | 'iter' | 'final' | 'notify'

3) FunValCheck {'off'} | 'on' GradObj 'on' | {'off'} Jacobian 'on' | {'off'}

4) LargeScale 'on' | 'off'. The default for fsolve is 'off'.

The default for all other functions that provide a large-scale algorithm is 'on'

算法函数默认都是 on,开启大规模算法

5) MaxFunEvals Positive integer 正整数

6) MaxIter Positive integer 正整数

7) OutputFcn Specify a user-defined function that an optimization function calls at each iteration. See Output Function.

8) TolCon Positive scalar

9) TolFun Positive scalar

10) TolX Positive scalar

11) TypicalX Vector of all ones

(2) Optimization options used by large-scale algorithms only

仅限大规模计算使用的参数设置

1) Hessian 'on' | {'off'}

2) HessMultFunction | {[]}

3) HessPattern Sparse matrix | {sparse matrix of all ones}

4) InitialHessMatrix {'identity'} | 'scaled-identity' | 'user-supplied'

5) InitialHessType scalar | vector | {[]}

6) JacobMult Function | {[ ]}

7) JacobPattern Sparse matrix | {sparse matrix of all ones} 稀疏矩阵

8) MaxPCGIter Positive integer | {the greater of 1 and floor(n/2)} where n is the number of elements in x0, the starting point

9) PrecondBandWidth Positive integer | {0} | Inf

10) TolPCG Positive scalar | {0.1}

(3) Optimization options used by medium-scale algorithms only

仅限中等规模计算使用的参数设置

1) BranchStrategy 'mininfeas' | {'maxinfeas'}

2) DiffMaxChange Positive scalar | {1e-1}

3) DiffMinChange Positive scalar | {1e-8}

4) GoalsExactAchieve Positive scalar integer | {0}

5) GradConstr 'on' | {'off'}

6) HessUpdate {'bfgs'} | 'dfp' | 'steepdesc'

7) LevenbergMarquardt 'on' | {'off'}

8) LineSearchType 'cubicpoly' | {'quadcubic'}

9) MaxNodes Positive scalar | {1000 * NumberOfVariables}

10) MaxRLPIter Positive scalar | {100 * NumberOfVariables}

11) MaxSQPIter Positive integer

12) MaxTime Positive scalar | {7200} MeritFunction 'singleobj' | {'multiobj'}

13) MinAbsMax Positive scalar integer | {0}

14) NodeDisplayInterval Positive scalar | {20}

15) NodeSearchStrategy 'df' | {'bn'}

16) NonlEqnAlgorithm {'dogleg'} | 'lm' | 'gn', where 'lm' is Levenburg-Marquardt and 'gn' is Gauss-Newton.

17) RelLineSrchBnd Real nonnegative scalar | {[ ]}

18) RelLineSrchBndDuration Positive integer | {1}

19) Simplex When you set 'Simplex' to 'on' and 'LargeScale' to 'off', fmincon uses the simplex algorithm to solve a contrained linear programming problem.

20) TolRLPFun Positive scalar | {1e-6}

21) TolXInteger Positive scalar | {1e-8}

3. 实例演示

(1) this statement creates an optimization options structure called options in which the Display option is set to 'iter' and the TolFun option is set to 1e-8

输出算法迭代过程，目标函数误差控制范围 1e-8，即 0.00000001

设置方法为：options = optimset('Display','iter','TolFun',1e-8)

(2) This statement makes a copy of the options structure called options, changing the value of the TolX option and storing new values in optnew

变量 X 的误差控制范围修改为 1e-4，即 0.0001

设置方法为：optnew = optimset(options,'TolX',1e-4)；